U.S.N.A --- Trident Scholar Report; no. 363 (2008)

Capability Driven Robotic Swarms in Reconnaissance-Based Operations

By

MIDN 1/C Evan A. Barnes
United States Naval Academy
Annapolis, Maryland

_____
(Signature)

Certification of Adviser Approval

Associate Professor Bradley E. Bishop
Systems Engineering Department

_____
(Signature)

_____
(Date)

Acceptance for the Trident Scholar Committee

Professor Joyce E. Shade
Deputy Director of Research & Scholarship

_____
(Signature)

_____
(Date)

USNA-1531-2

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 074-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including g the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of the collection of information, including suggestions for reducing this burden to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATE COVERED
	2 May 2008	

4. TITLE AND SUBTITLE
Capability Driven Robotic Swarms in Reconnaissance-based Operations

5. FUNDING NUMBERS

6. AUTHOR(S)
Barnes, Evan A.

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)		8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)		10. SPONSORING/MONITORING AGENCY REPORT NUMBER
US Naval Academy Annapolis, MD 21402		Trident Scholar project report no. 363 (2008)

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT	12b. DISTRIBUTION CODE
This document has been approved for public release; its distribution is UNLIMITED.	

13. ABSTRACT - While 21st century surveillance techniques have started to utilize unmanned robotic platforms to gradually relieve many of the dangers and pressures placed on a human being, there is still a significant cost issue when addressing the overall effectiveness and efficiency. It is obvious that the unmanned robotic vehicle's ability to carry out surveillance missions is unparalleled by any human due to technological advances in electronic sensors. However, a single autonomous robot faces many limitations which decrease its cost efficiency and prevent it from being the best choice for various missions. On the other hand, a group of robots working together can remove many of the limitations imposed on one single unit. However, controlling a group of robots is inherently more complex than controlling just one. Initial controllers focused on the swarms mean position and variance in order to control its individual members and direct the entire swarm. This approach was successful in directing a swarm to a certain location; however, the swarm units were not always in the best position to use their individual capabilities. Since there is no way to control each individual unit's position without specific coding, the swarm may be in the correct location but some of the units may be rendered useless due to their position to the target point. This is one of the major concerns for using a mean/variance based controller in reconnaissance operations. This standard controller is better used for formation control of relative unit position based operations .This project developed new methodologies for optimizing the performance of a heterogeneous swarm in reconnaissance based operations. Using a hybrid systems-behavior control theory, a swarm of small, non-holonomic, robotic vehicles with
(cont on p.2)

14. SUBJECT TERMS	15. NUMBER OF PAGES
Robotic swarm, capability, navigational capability, deliverable capability, hybrid swarm controller	149
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT

Abstract

While 21st century surveillance techniques have started to utilize unmanned robotic platforms to gradually relieve many of the dangers and pressures placed on a human being, there is still a significant cost issue when addressing the overall effectiveness and efficiency. It is obvious that the unmanned robotic vehicle's ability to carry out surveillance missions is unparalleled by any human due to technological advances in electronic sensors. However, a single autonomous robot faces many limitations which decrease its cost efficiency and prevent it from being the best choice for various missions. On the other hand, a group of robots working together can remove many of the limitations imposed on one single unit. However, controlling a group of robots is inherently more complex than controlling just one.

Initial controllers focused on the swarms mean position and variance in order to control its individual members and direct the entire swarm. This approach was successful in directing a swarm to a certain location; however, the swarm units were not always in the best position to use their individual capabilities. Since there is no way to control each individual unit's position without specific coding, the swarm may be in the correct location but some of the units may be rendered useless due to their position to the target point. This is one of the major concerns for using a mean/variance based controller in reconnaissance operations. This standard controller is better used for formation control of relative unit position based operations.

This project developed new methodologies for optimizing the performance of a heterogeneous swarm in reconnaissance based operations. Using a hybrid systems-behavior control theory, a swarm of small, non-holonomic, robotic vehicles with potentially heterogeneous abilities was able to autonomously explore an unknown area and maximize their performance in relation to some mission specified on the sensor modality. Using capability

functions to help direct the individual units of a swarm, the swarm automatically achieved its overall objective and provided more useful information on a target location. The first part of this project focused on systems design and control. At the end of this phase, definition and design of the capability functions, primary and secondary objectives, a suitable test swarm and an appropriate test bed were made and the created controller was simulated in a variety of environments. The Second part of the project focused on hardware implementation, and resulted in a real life demonstration of the designed control methodologies utilizing Khepera II Mobile Robots in a constructed environment. The robots behaved according to their individual capabilities and worked together in order to optimize the swarms overall performance.

Keywords:

- Robotic Swarm
- Capability
- Navigational Capability
- Deliverable Capability
- Hybrid Swarm Controller

Acknowledgements:

I would like to thank the Trident Scholar Committee as well as the staff and faculty of the Weapons and Systems Engineering Department for all of their assistance.

Table of Contents

Table of Figures

Table of Equations

1. Introduction

The goal of this project was to explore how to control a heterogeneous swarm of robots in a reconnaissance setting, utilizing their individual strengths to enhance efficiency and mission performance. Most current swarm control algorithms are designed for homogeneous swarms, both in kinematics/dynamics and in additional capabilities (sensing, communication, etc.). However, for this project heterogeneous swarms comprised of units with varying kinematic constraints (varying speed capabilities) as well as sensing capabilities were considered. For example, some of the robots had low-resolution wide-angle cameras while others were equipped with narrow field-of-view vision systems with a longer range. It is differences like these that require the individual robots to behave differently while still cooperating to accomplish the overall swarm goal. Since the swarm was expected to perform autonomously, the robots had to be able to interact with their environment based upon their capabilities. By optimizing the use of each robot's individual capability, the swarm could efficiently accomplish many different missions or objectives.

In this work, individual units in a swarm that possessed some capability that was to be delivered to a target are considered. Capability may mean sensing of the target (vision, audio, chemical, etc.), weapons fire (machine guns, grenades, smoke bombs, etc.) or something as simple as lighting. The framework developed was independent of the exact capability to be delivered, although the focus was on sensing for a reconnaissance-style mission.

While the primary objective of this work was for a swarm to deliver appropriate capability to various locations, it had to also meet certain secondary objectives in order to be a viable option for a real mission. Table 1 displays a set of relevant secondary objectives.

Primary Objectives	Secondary Objectives
• Deliver appropriate capabilities to target location(s).	• Avoid collisions with obstacles • Avoid collision between individual robots • Avoid collision with objective location • Achieve Line of Sight Capability • Minimize the effects of local minima • Automatic resource allocation • Efficient use of power • Low response time for emerging needs • Capable of flexibility in specification of mission and objectives

Table 1: Primary and secondary objectives for the swarm controller

All of the objectives in Table 1 had to be met in order for the swarm to be a success. It was apparent that the primary objective pertained to the entire swarm as a single entity, whereas some of the secondary objectives concerned each individual unit's interaction with its surroundings and other members of the swarm. Without meeting the primary objectives, the swarm would not serve a purpose. However, it is the secondary objectives which must be met in order to make the swarm an efficient choice for any given mission.

2. Background

The concept of swarm control has been widely investigated by researchers due to a swarm's capability to complete a task with greater cost efficiency than a single high-priced robotic platform. In applications from Mine Countermeasures to Emergency Search and Rescue

(SAR), swarm technology has the potential to effectively complete many missions deemed too hazardous for human beings, or too costly or risky for one single robotic platform. Single robots performing missions such as search and rescue, reconnaissance or surveillance often encounter problems dealing with line of sight, which can limit their effectiveness.[1] While this reference is primarily concerned with LOS-based communications, the basic problem is that capability (of any sort) may be blocked by various types of obstacles. The basic problem is shown in Figure 1, below. A swarm of robots is not necessarily limited by this issue due to its ability to surround obstacles or points of interest and provide multiple points-of-view on a target – offering increased mission versatility and capability.

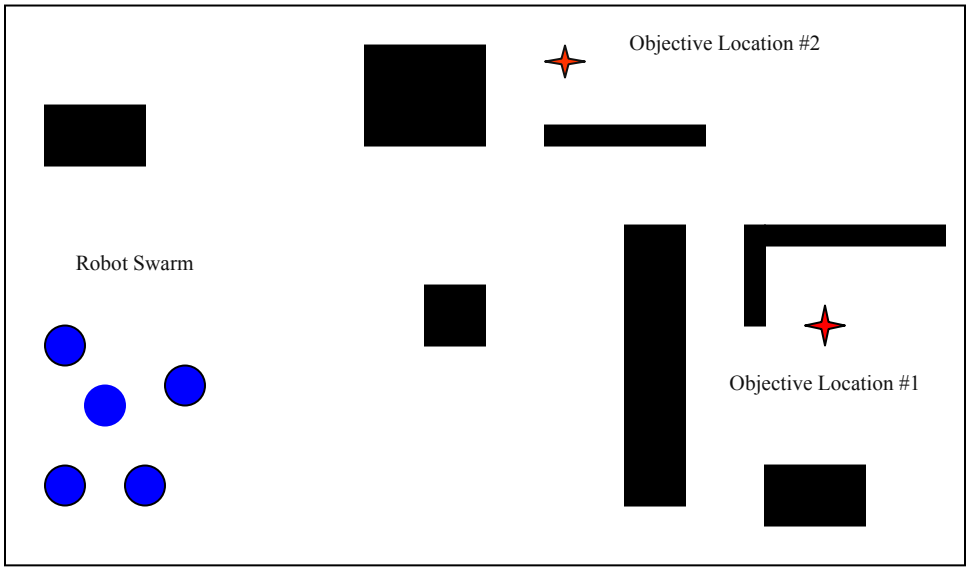

Figure 1: Sample Multi-Target Environment

A swarm's ability to accomplish a certain task in an infinite number of ways (depending on the parameterization of the problem) is the backbone of its performance capacity. This is possible due to the concept of redundancy. In the realm of robotic control, a redundant manipulator possesses more joint space degrees of freedom than task variables.[2] For example, if the goal of a swarm is to move its mean position to a certain location on a 2-Dimensional surface

and maintain a specific variance pertaining to its members, it would require a minimum of 4 degrees of freedom. Each degree of freedom would pertain to one of the task variables: mean x-position, mean y-position, and swarm variance in each dimension. However a swarm of 10 simple robots on a 2D surface possesses at least 20 degrees of freedom. This allows it to assume an infinite number of formations as long as the swarms mean position and variance match the desired values. As one robot moves, another robot can adjust to counteract any change to the swarms position or variance.

While the potential of a swarm of robots is virtually unlimited, there are still many questions concerning the proper control methodologies and swarm styles. Swarm robotics poses a difficult challenge with respect to control theory and data retrieval. In swarm robotics, there are two broad categories into which most swarm control methods fall: "unintelligent" and "intelligent" swarms[3]. These types of swarms differ in their control methodologies and procedure for accomplishing a given task. "Unintelligent" swarms are based on the principle of trial and error. Through sheer repetition and unlimited failure, an "unintelligent" swarm continues to operate until it has accomplished a task. An example of an "unintelligent" swarm could be a given number of small robots that are designed to move in one direction until they encounter an obstacle or find their target. If the robots run into an obstacle, they turn and go another direction, typically at random. While this procedure would eventually accomplish the objective of locating a target in a confined area, this approach may prove much less efficient than other methods. However, due to the swarm's ability to cover a widespread area while dealing with many failures, there are a few situations where an "unintelligent" swarm is desired.[4]

"Intelligent" swarms on the other hand, employ various sensors that allow them to cooperate, coordinate motion and communicate with other members of the swarm. The members

of an "intelligent" swarm work together to accomplish a specific objective, rather than acting as individual entities in a large group. They are typically able to sense each other and their surroundings and position themselves accordingly by using various means of control. Intelligent swarms utilize some form of overall control, whether it be centralized or decentralized.[5] With centralized control, all of the members of the swarm report back to either one robot or computer that computes desired motions for every unit simultaneously, in a coordinated fashion. Decentralized control, however, demands that the entities in a swarm be able to think for themselves while remaining a part of the swarm. Each unit uses data collected from the environment and potentially from the other units to compute its own desired motion.

There are, however, slight adaptations of centralized and decentralized control which result in a more effective and efficient swarm capabilities; systems-theoretic and behavior-based methods. Systems-theoretic techniques guarantee strong system performance, and their control methodologies are more easily understood. They involve the direct coding or command of all units in a swarm based on a desired system response for modeled stimuli. Therefore, every unit in the swarm is directly controlled to guarantee a specific outcome. However, in order for this to be a viable option for a certain mission, all of the information pertaining to the environment must be known or at a minimum highly predictable. Alternatively, behavior-based control approaches permit a swarm to automatically adapt to an unknown environment, and the control method is simpler in nature. The outcome of behavior-based controllers is inherently unpredictable in nature and often gives rise to interesting emergent outcomes as a result of the swarm's interaction with its surroundings.[6] The tradeoff between these two approaches involves guaranteed performance versus system flexibility.

Another important area of interest pertaining to robot swarms involves the design of each individual unit. While most vehicles are non-holonomic in nature, it is often easier to assume the swarm entities are holonomic which simplifies the simulation and allows us to avoid having to account for kinematic constraints. Holonomicity (for mobile robots) effectively refers to the relationship between controllable and total degrees of freedom for a specific platform. For example, a car is non-holonomic because although it can move laterally (e.g. parallel parking) it does not have the control capabilities to do as directly as it must first move in a forward motion and then backward motion. However, it is possible to model a differentially-driven robot system as holonomic as long as the state to be controlled is defined as a position (2D) off of the common wheel axis.[7] This can be seen in Figure 2. The robot in Figure 3, on the other hand, must be modeled as a non-holonomic system because the front wheel must first rotate towards the desired direction of motion. At any given instant, the robot has directions of motion that it cannot achieve without preceding gross unit motions.

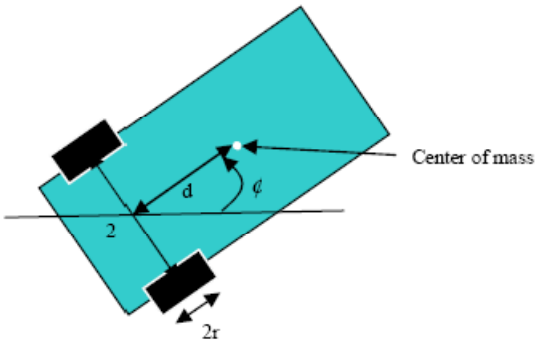

Figure 2: Standard non-holonomic vehicle that can be simulated as a holonomic platform.[8]

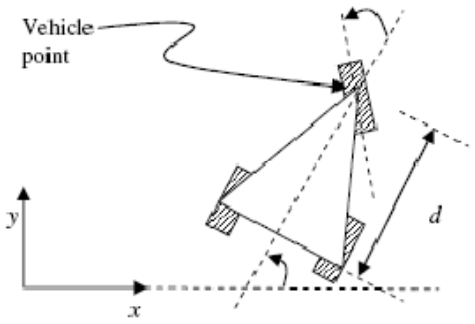

Figure 3: Non-holonomic vehicle.

It is obvious that there are potentially an unimaginable amount of uses for robotics swarms as well as many different control theories to go with them. Published research on such uses and control theories, which have been researched but not utilized in this report, can be seen in the References section along with works that have been used and cited.

3. Capability Functions

A. Introduction

This controller was based upon the unit's ability to deliver capability on target. Therefore, the mathematical expression for each different capability was first defined. This controller was able to control a swarm of robots with different capabilities, as long as there was a mathematical representation for that capability. This helped to promote the desired level of flexibility for my controller so that it could be used in a variety of operations. For this project, the main type of sensor used was a pin-hole lens system.

B. Pin-Hole Lens System

Since the controller was based upon the overall capability delivered to a certain target location, what was first developed were the individual capability functions for each of the swarm units. These capability functions depended on the specific sensing capability of the individual unit and were unique to each sensor. The first step was developing the capability function for a short range vision system. One of the first assumptions made was that a pin-hole camera lens systems was used. This system was the most basic representation of a vision system and therefore the simplest to model. Figure 4 is a model of a pin-hole lens system where one can see by inspection how the size of the object in the image plane is a function of its actual size, focal length and its actual distance from the camera. Meanwhile, Figure 5 provides a simplified model relating the position in the camera frame to that of the image frame.[9]

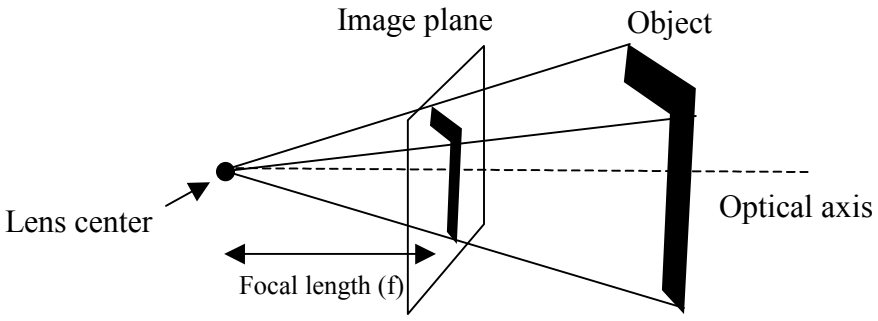

Figure 4: Pin-Hole Lens System

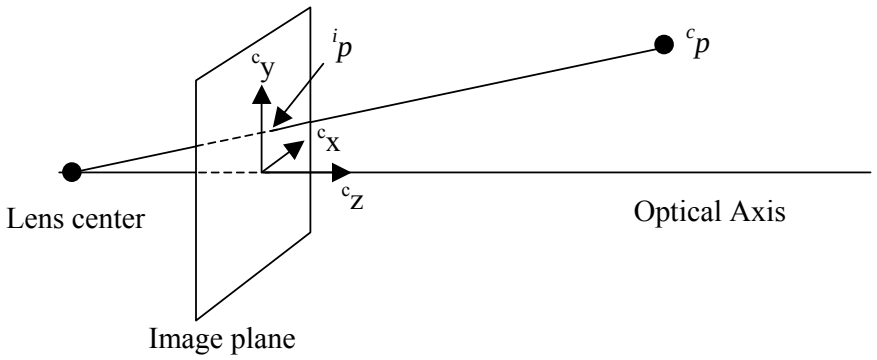

Figure 5: Variable Model

Based upon the model seen in Figure 5, a mathematical formula relating distances in the camera frame to the image frame can be derived.

$$ {}^{i}_{x}p = \frac{f * {}^{c}_{x}p}{{}^{c}_{z}p + f} $$

Equation 3.1

$$ D = {}^{c}_{z}p + f $$

Equation 3.2

From these equations, it became obvious that the size of an object in real life compared to the size in the image frame was just proportional to the focal length of the lens system divided by the distance of the object from the camera lens.

C. Lens and System Parameters

Using a standard 35 mm lens system, my first step was to develop an appropriate means to determine the required distance from target. Since the goal of the system was to return information on a certain location, it was determined that the appropriate unit of measurement would be the pixel resolution (pixels/meter) at a certain distance from the lens system. The desired resolution was set to 400 pixels per meter or .25 cm per pixel. This gave a balance between clarity in image and flexibility in movement. The robots would then be able to maintain a reasonable distance while still being able to return usable information on the objective location. In the image plane a resolution of 300 ppi (pixels per inch) was desired. This is a common standard for quality LCD image resolution and the pixel width is equal to 0.0846 mm/pixel.

Once the basic controller parameters were decided, Equation 3.1 and Equation 3.2 were used to create a basic proportional comparison where *D* was equal to the distance from the lens system, *f* was the focal length of the lens system, *N* was the number of pixels per meter of target area and *w* was the pixel width in the image plane.

$$N = \frac{f}{D * w}$$

Equation 3.3

From this equation a visual representation of how the pixel resolution relates to the distance of an object for a variety of focal lengths was created as seen in Figure 6.

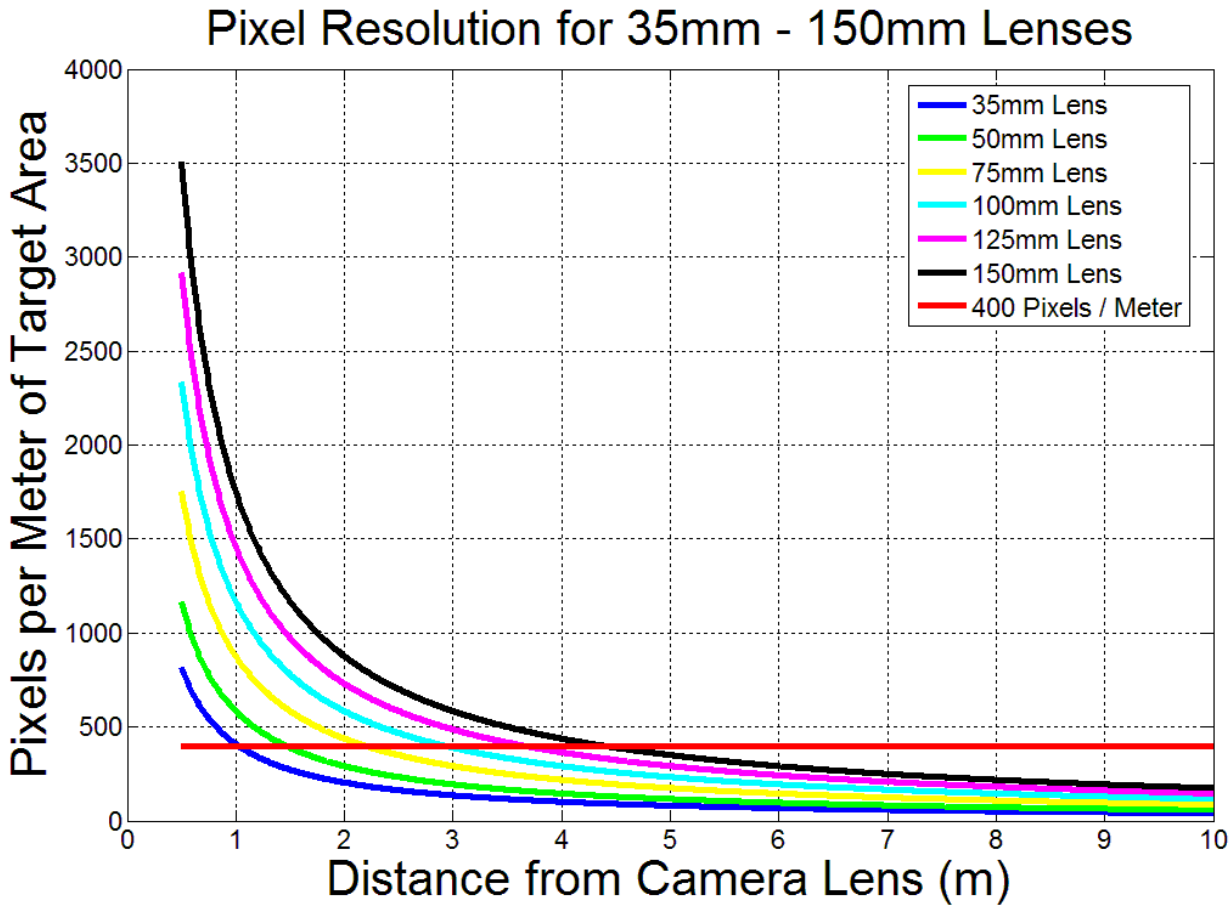

Figure 6: Visual representation of the pixel resolution for 35 mm - 150 mm lenses

The intersection between the various plots and the horizontal line representing our desired pixel resolution of 400 pixels per meter of target area provides us with the desired distance of the object from the lens system. This distance was defined as the point at which our camera system was delivering 100% capability to the target location. This distance was *k* and it was calculated by the following equation.

$$k = \frac{f}{N_{opt} * w}$$

Equation 3.4

D. Initial Capability Functions

Now that a desired distance was established, development began on the overall capability equations that represent the respective vision systems. Beginning with a simple 2D equation, which was solely a function of object distance, a starting point was provided to develop the final equation. Using the previously determined *k* value, the following capability equation was generated. It was necessary to have two separate equations based on object distance because once an object was less than the desired distance away, pixel resolution became less of an issue and the ability of the camera to focus on that object became the presiding limitation. The combination of these limited equations resulted in one differentially smooth curve.

$$C_j = f(x) = \begin{cases} \sin\left(\frac{\pi}{2} * \frac{D}{k}\right)^M, & D < k \\ \sin\left(\frac{\pi}{2} * \frac{N}{N_{Opt}}\right), & D \geq k \end{cases}$$

Equation 3.5

In this equation C represented the overall capability at that distance, while M was a gain variable unique to each system, N_{opt} was the desired pixel resolution (pixels per meter of target area), and k was the distance at which N_{opt} was achieved. The 2-Dimensional plot of C for a 35 mm lens can be seen in Figure 7 while Figure 8 shows the capability C for a 100 mm lens.

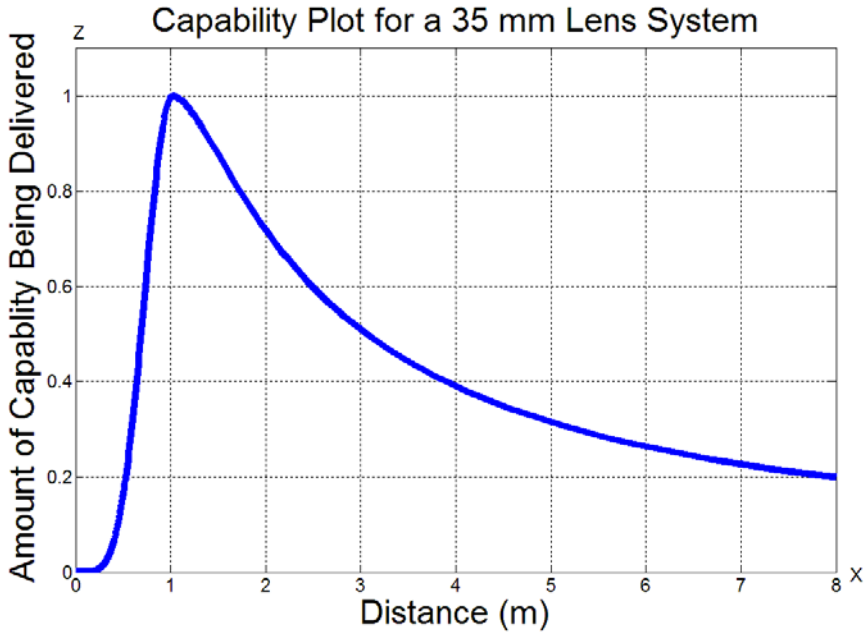

Figure 7: Capability being delivered on target as a function of distance for a 35 mm lens.

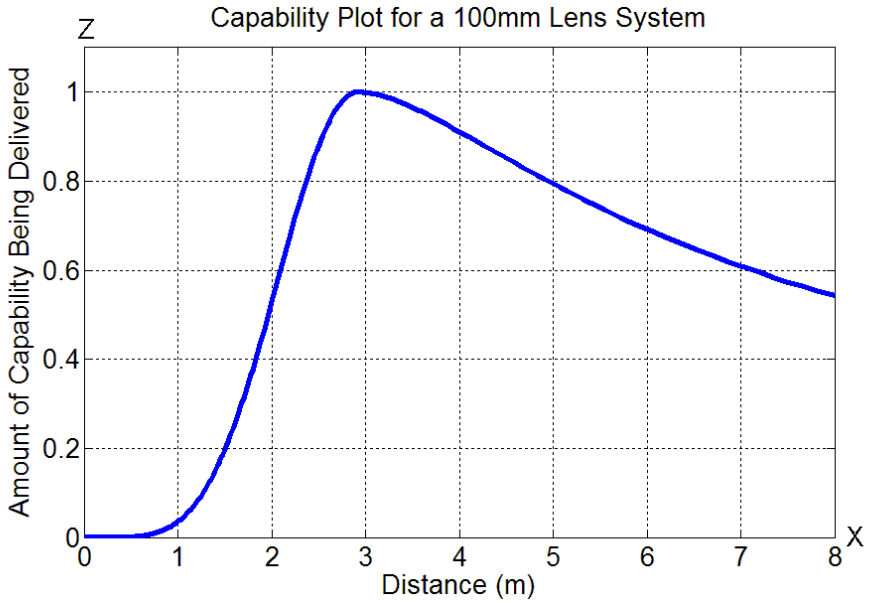

Figure 8: Capability being delivered on target as a function of distance for a 100 mm lens.

E. "Capability Volcanoes"

Once an equation of capability with respect to distance was derived, the next step was to find the capability of a robot in a 3-D space. Assuming that capability was a strict function of distance from the lens center and applying Equation 3.5 to the 3-D space resulted in a circular function of capability called the "Capability Volcano". This was a simplistic but differentiable view of the capability that can be delivered by a lens system. This was the first step toward using the capability to direct movement of the swarm, and was useful for visualization. Various points of view of the "Capability Volcano" for 35 mm and 100 mm Lenses can be seen in Figure 9 through Figure 12.

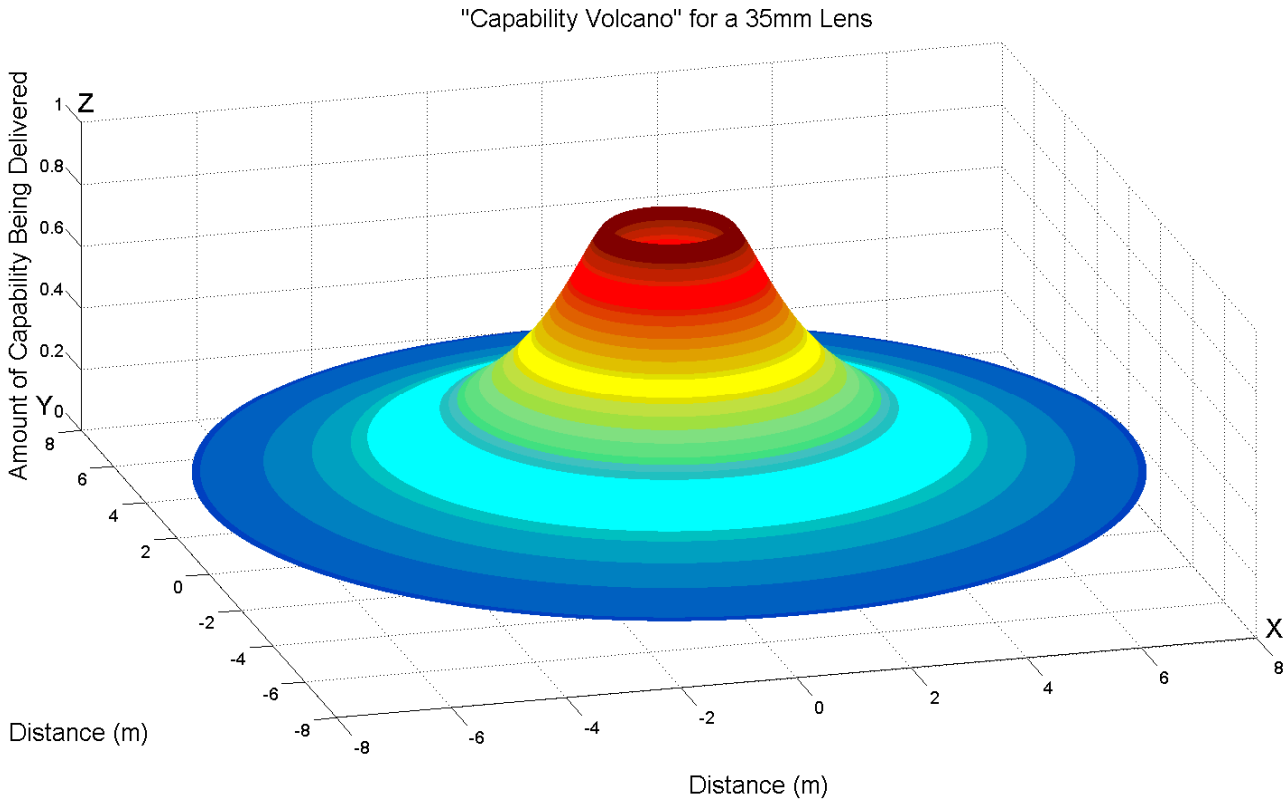

Figure 9: 3D representation of the capability delivered by a 35 mm Lens

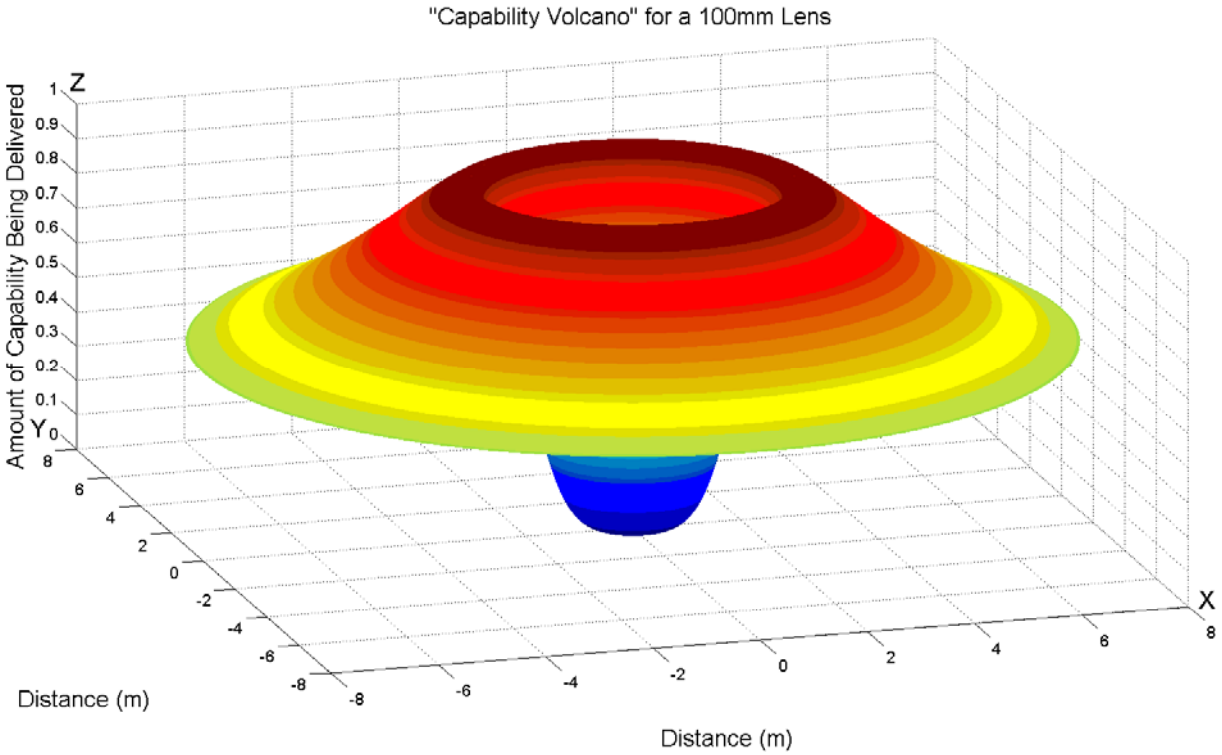

Figure 10: 3D representation of the capability delivered by a 100 mm Lens

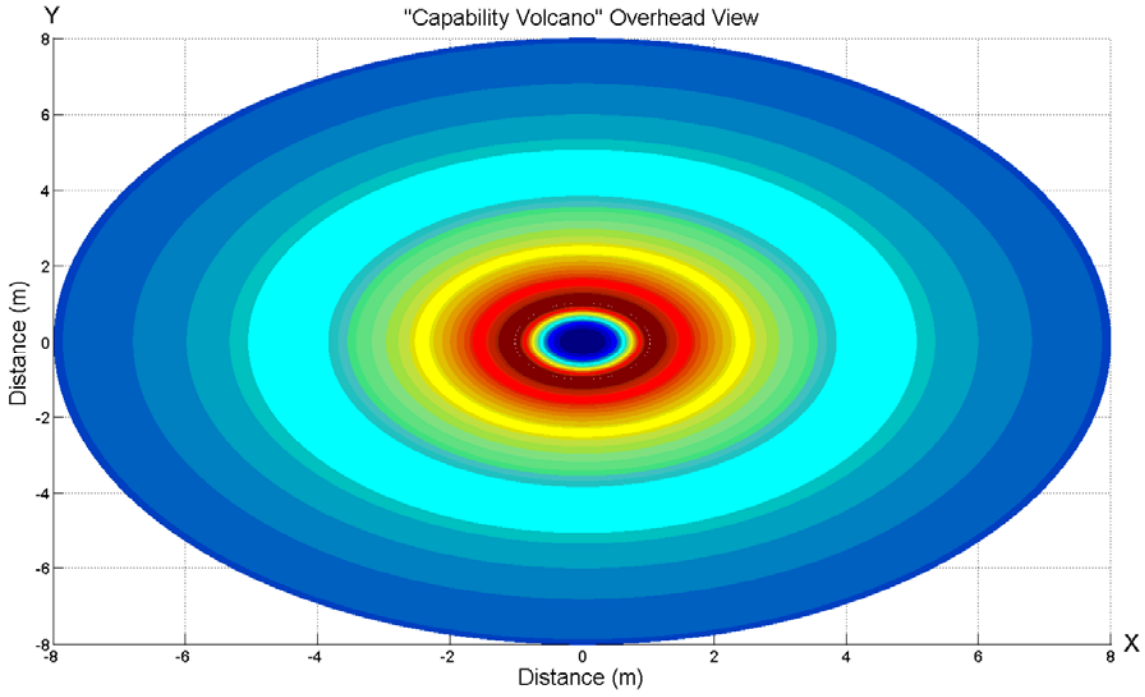

Figure 11: Overhead view of the "Capability Volcano" created by a 35 mm Lens.

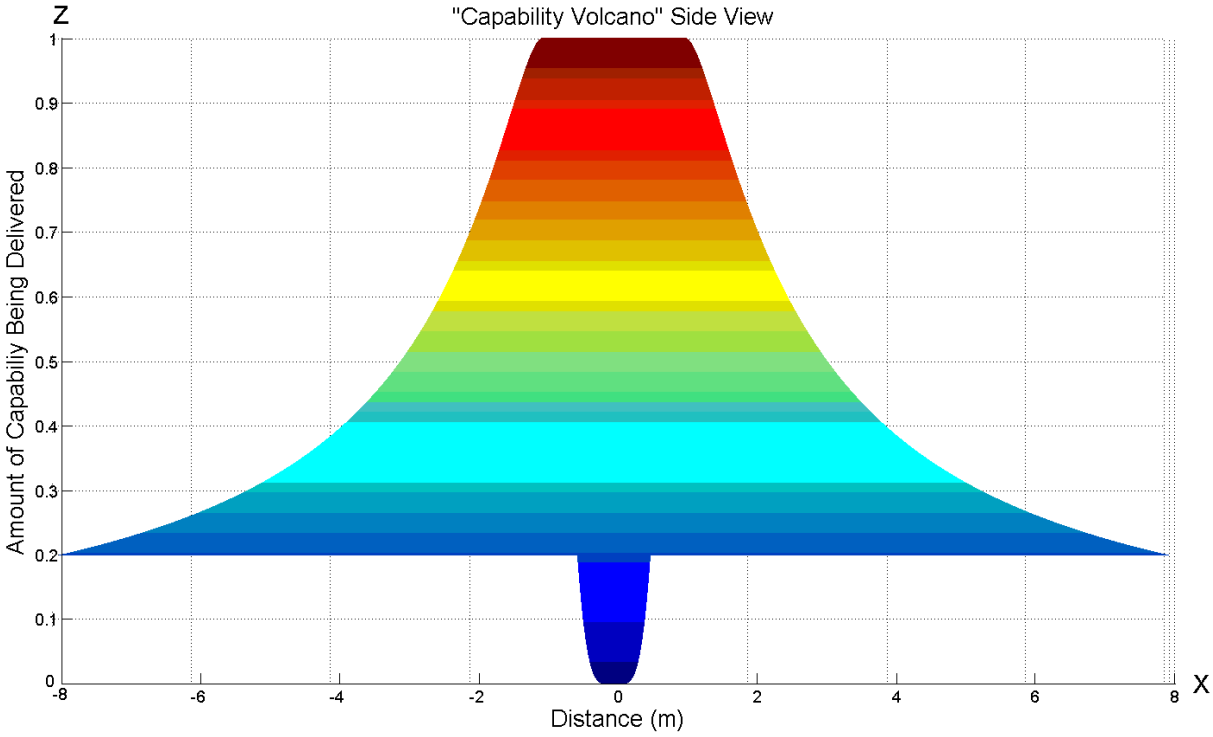

Figure 12: Side view of the "Capability Volcano" created by a 35 mm Lens

F. Deliverable Capability

Even though this was not necessarily an accurate representation of actual capability delivery due to the limits of field of view, it provided a differentially smooth function required to direct the individual unit's motion. When calculating the capability delivered on target, one must threshold the capability function to account for field of view as well as the focusing ability of the lens. This thresholded value was the actual capability on target. By simply manipulating the gains as well as the other variables, one is able to develop unique functions for each individual sensor. As one can see, for all locations outside the field of view of the lens, the capability being delivered was 0. This plot was called the "Deliverable Capability". Because the entire curve was not differentially smooth one was forced to use the Navigation Capability to control

motion and use the Deliverable Capability to determine mission success. It was assumed for Figure 13 through Figure 22 that the camera was pointed along the y-axis.

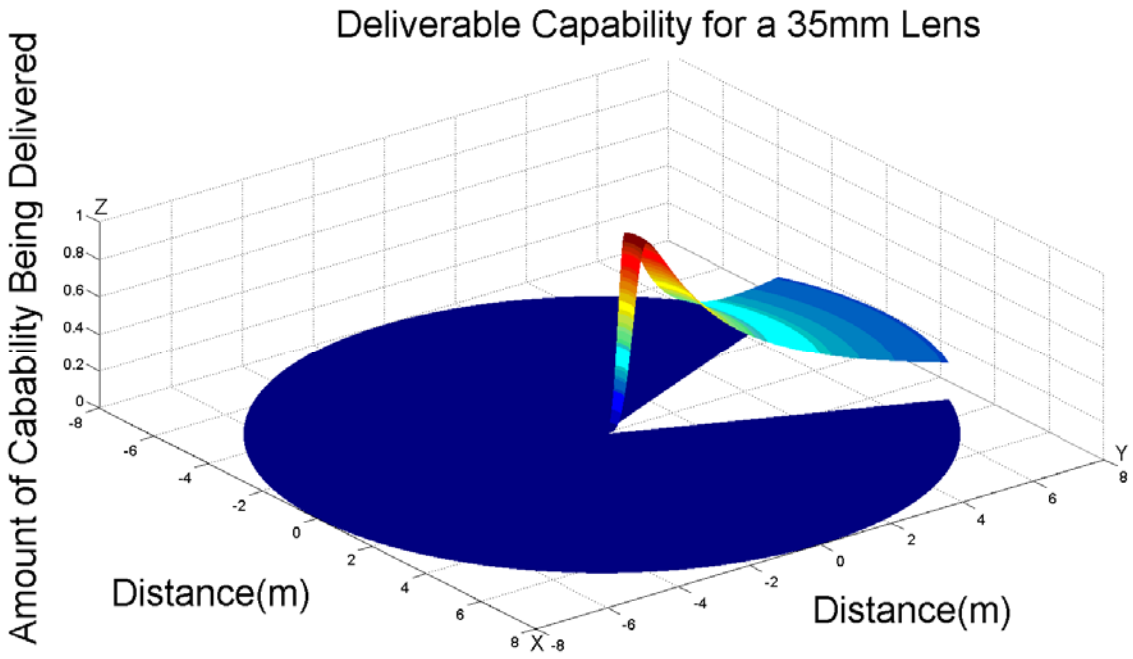

Figure 13: Deliverable capability for a 35 mm lens with a field of view of 45°

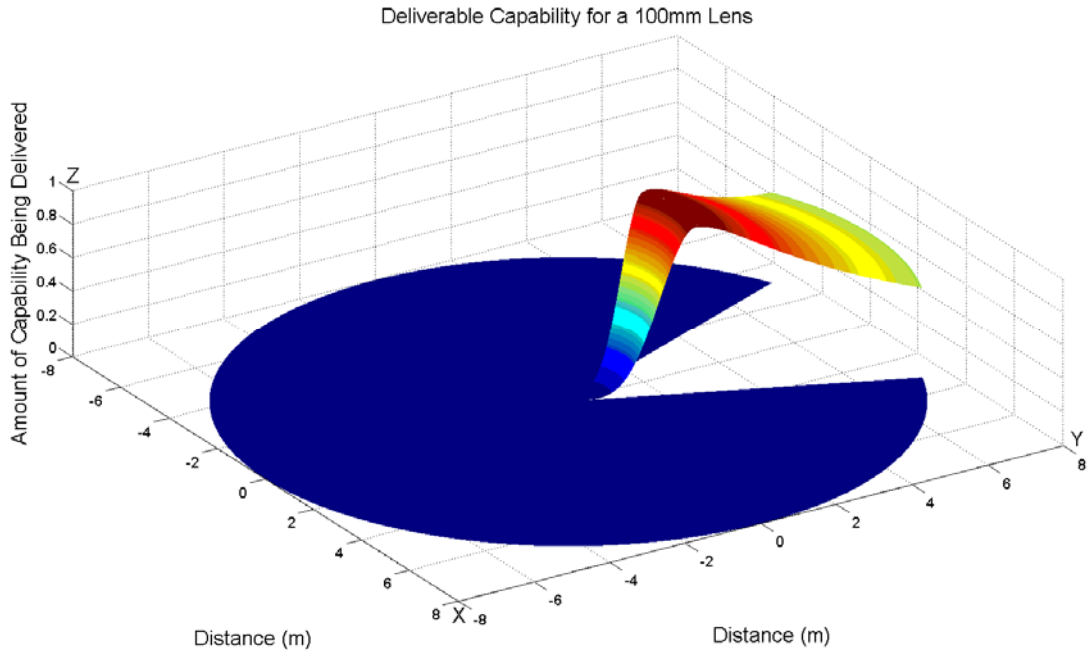

Figure 14: Deliverable capability for a 100 mm lens with a field of view of 45°

G. Navigational Capability

The "capability volcano" was inherently inaccurate because in real life the camera was only able to see what was in its field of view, which was rarely 360°. Therefore a limiting factor was added to the initial equation which adjusted the capability based upon the difference from the robots current heading and the actual bearing to target. The resulting equation is below:

$$C_j = f(x) = \begin{cases} \sin(\frac{\pi}{2} * \frac{D}{k})^M * 1/10^{\sin(\frac{\theta-\alpha}{2})^4}, & D < k \\ \sin(\frac{\pi}{2} * \frac{N}{N_{Opt}}) * 1/10^{\sin(\frac{\theta-\alpha}{2})^4}, & D \geq k \end{cases}$$

Equation 3.6

This provided a 3-D differentially smooth function covering 360° field of view and a distance of 0 to ∞ as seen in Figure 15 through Figure 22. This capability was the "Navigational Capability" for the individual robot. The reason for this was that the differentially smooth curve was used to calculate gradient responses to direct the desired motion of the robot to accomplish the primary task. As one can see, it accounted for 360° field of view and distances of 0 to ∞. These plots however were limited in distance from 0 to 8 meters.

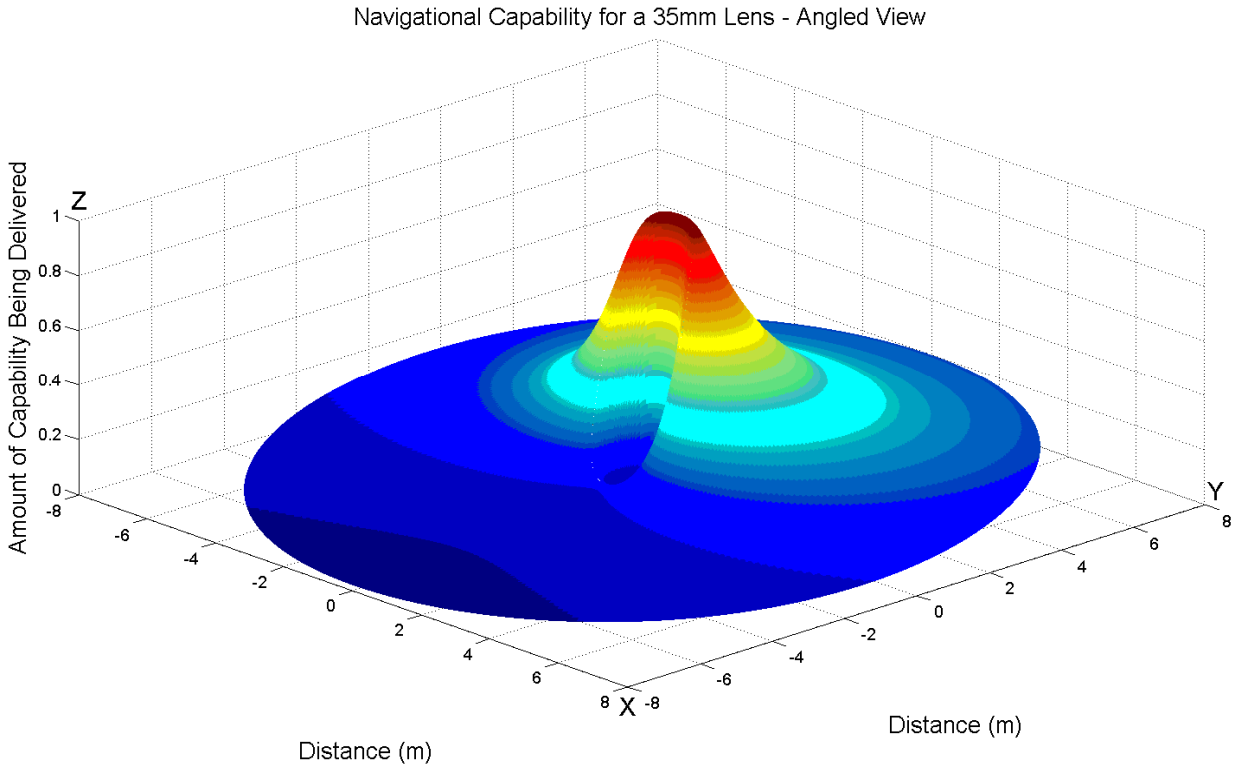

Figure 15: Navigational capability for a 35 mm lens.

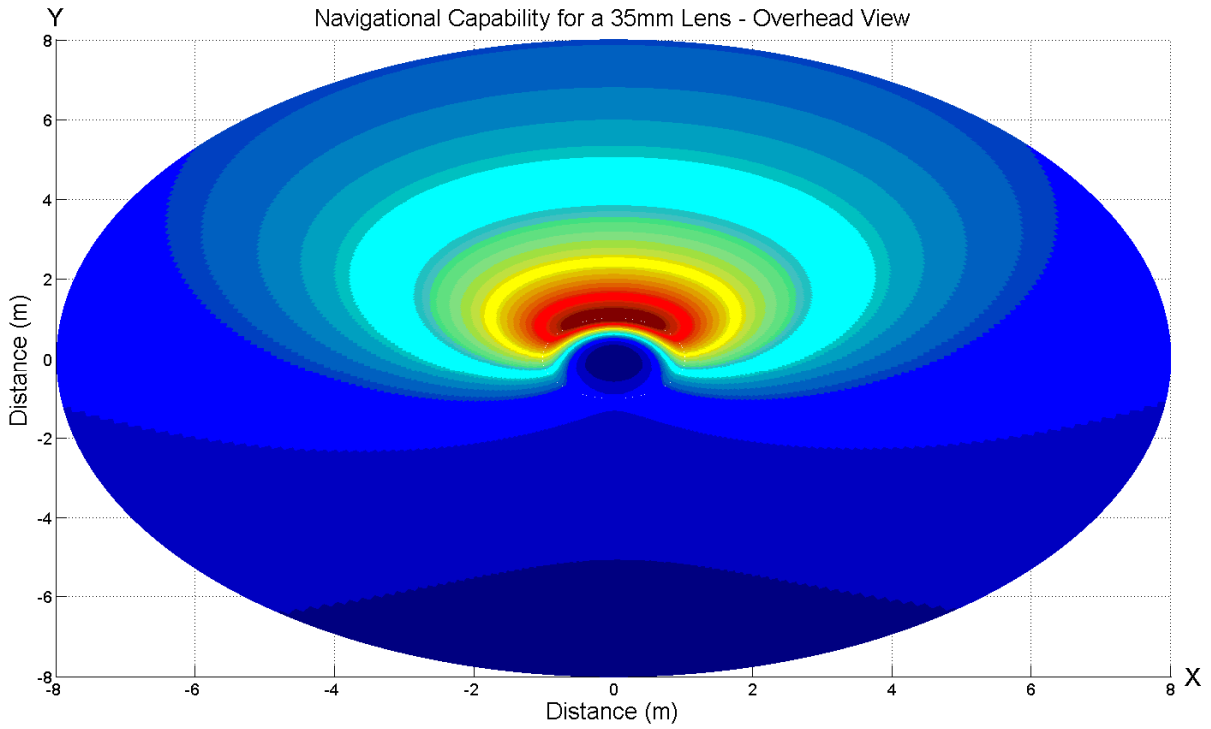

Figure 16: Overhead view of the navigational capability for a 35 mm lens.

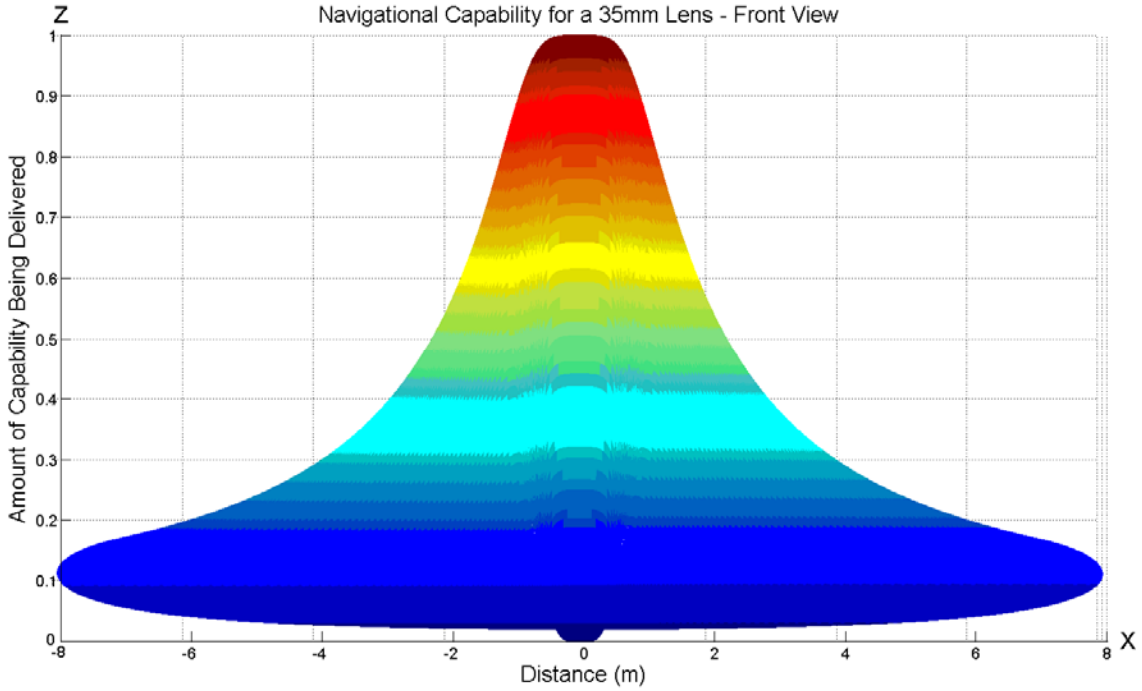

Figure 17: Front view of the navigational capability for 35 mm lens.

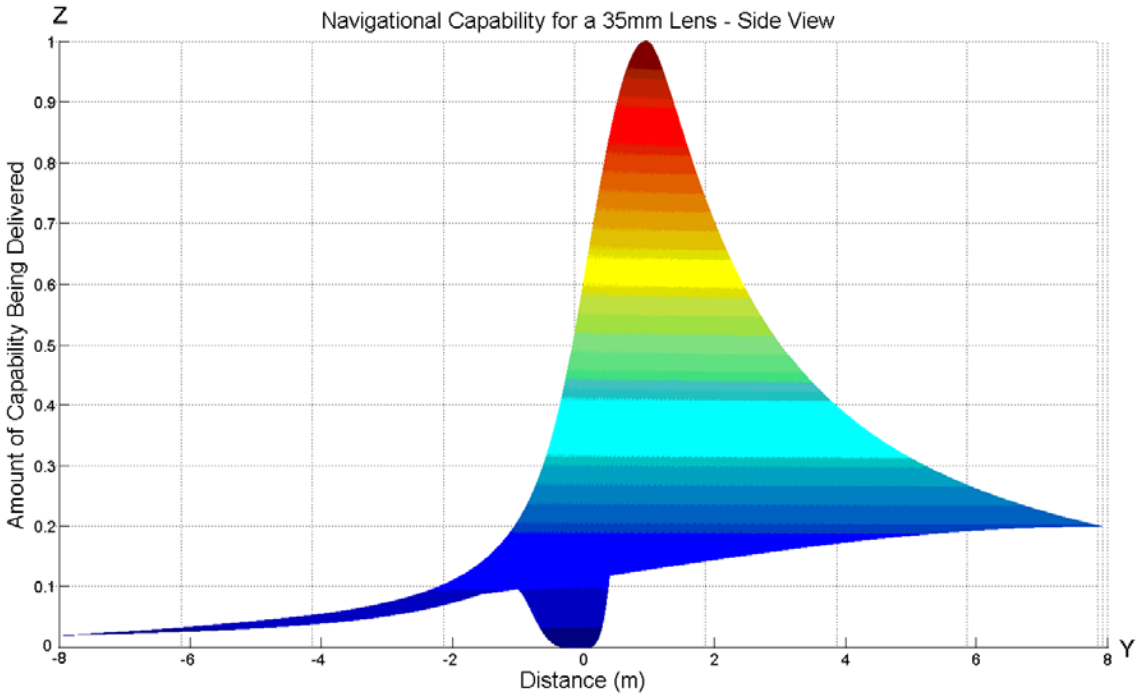

Figure 18: Side view of the navigational capability for a 35 mm lens.

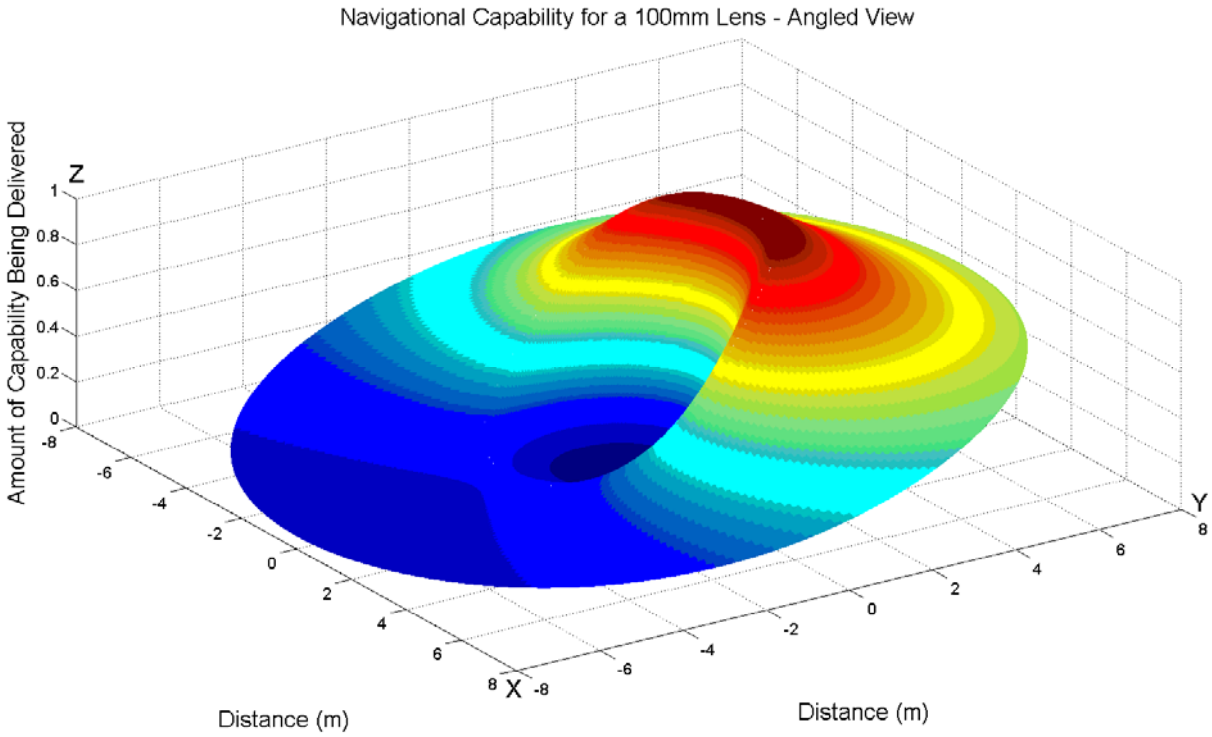

Figure 19: Navigational capability of 100 mm lens.

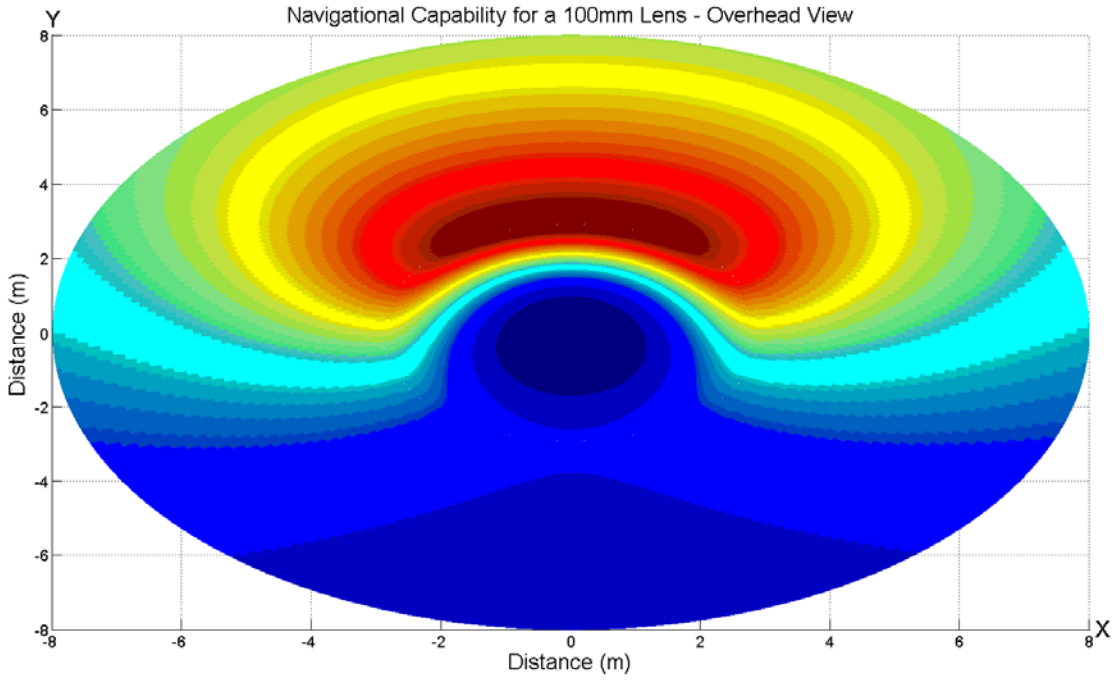

Figure 20: Overhead view of the navigational capability for a 100 mm lens.

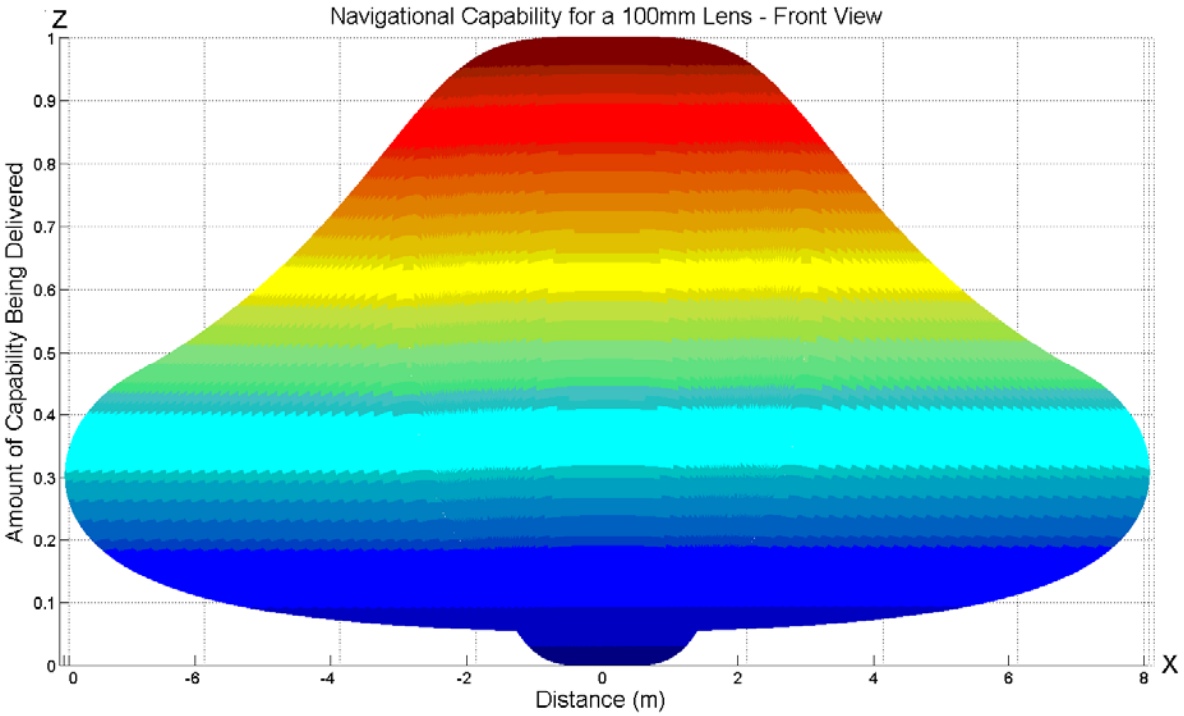

Figure 21: Front view of the navigational capability for a 100 mm lens.

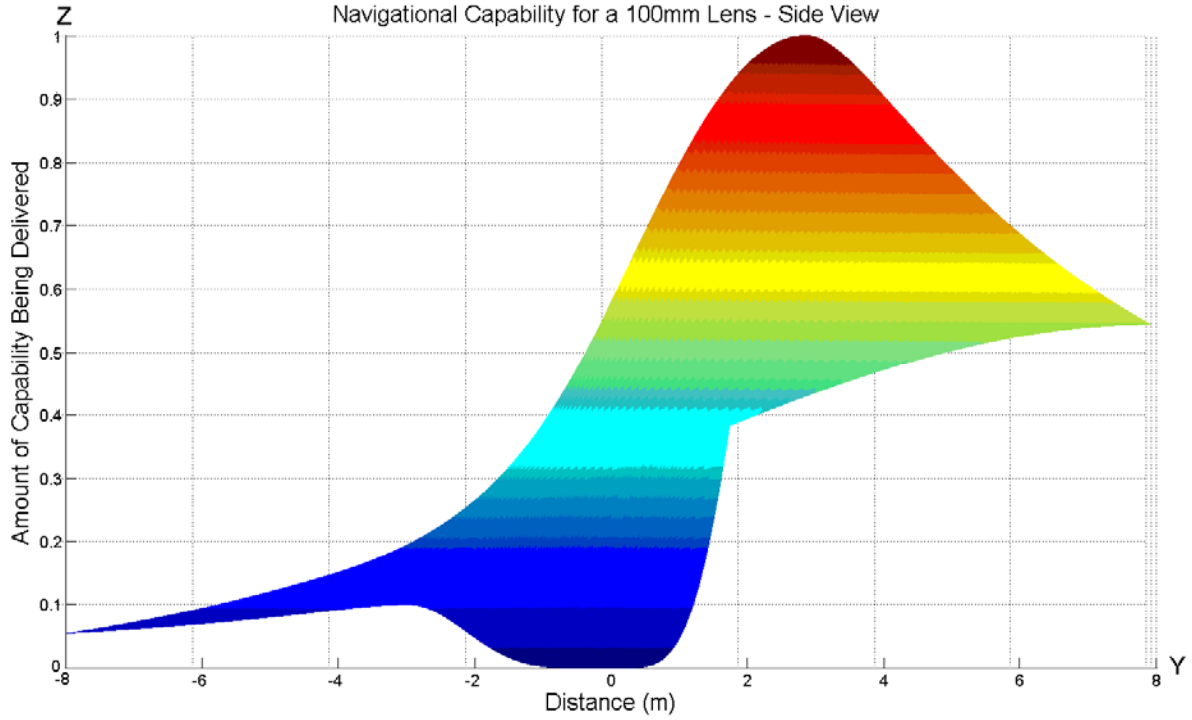

Figure 22: Side view of the navigational capability for a 100 mm lens.

As one can see, the peak capability was at a greater distance from the lens center for the 100 mm lens compared to the 35 mm lens. Once again, this plot was limited to a distance of 0 to 8 meters; however, capability was calculated to an infinite distance.

4. Initial Swarm Controller

A. Controller Matrices

Once one was able to quantify the capability of a robot at any point in space, one could begin work on the initial controller for the entire swarm. The robot Cartesian positions and camera orientation were initially stored in an Nx3 matrix in which N was the number of robots in the swarm and the target locations were stored in an Mx2 matrix where M was the number of targets. The initial locations were decided upon by the user creating the environment. This required a certain level of flexibility because the robots had to be able to adapt. The initial matrices are below.

$$Robot = \begin{bmatrix} X_{R1} & Y_{R1} & \theta_{R1} \\ X_{R2} & Y_{R2} & \theta_{R2} \\ \vdots & \vdots & \vdots \\ X_{RN} & Y_{RN} & \theta_{RN} \end{bmatrix}$$

Equation 4.1

$$Target = \begin{bmatrix} X_{T1} & Y_{T1} \\ X_{T2} & Y_{T2} \\ \vdots & \vdots \\ X_{TM} & Y_{TM} \end{bmatrix}$$

Equation 4.2

Once the initial environment was established, there were certain parameters that had to be calculated for each unit based upon the particular unit's sensor characteristics. Depending on the focal lengths assigned to each unit's lens system, the respective *k* value for that unit was calculated based upon Equation 3.4. The desired direction of motion for each unit was also calculated and this was represented by β. Each unit may also possess unique locomotive constraints. In this case, different unit maximum speeds were used to create a heterogeneous swarm. After assigning the individual units' their respective characteristics, the robot matrix had changed from an Nx3 to an Nx7 matrix. This was so the controller could access the individual robots information at any point throughout the simulation.

$$\text{Robot} = \begin{bmatrix} X_{R1} & Y_{R1} & \theta_{R1} & \beta_{R1}^{d} & f_{R1} & k_{R1} & Max\_Speed_{R1} \\ X_{R2} & Y_{R2} & \theta_{R2} & \beta_{R2}^{d} & f_{R2} & k_{R2} & Max\_Speed_{R2} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ X_{RN} & Y_{RN} & \theta_{RN} & \beta_{RN}^{d} & f_{RN} & k_{RN} & Max\_Speed_{RN} \end{bmatrix}$$

Equation 4.3

B. Jacobian Matrix

Since the controller could now track every unit's position and individual characteristics, the units were able to act accordingly in order to complete the primary objective. For the primary objective the controller used a Systems-Theoretic Controller which provided a certain level of assurance for mission success if it was possible. The first step was to develop a Jacobian matrix. This was the primary task controller for the swarm and provided directional calculations to the individual members of the swarm based upon the capability gradient. The Jacobian matrix was calculated by computing the partial derivative of each navigational capability function with

respect to its individual variables. The matrix represented how a change in any one of the robots poses would affect the capability at a target point.

$$J = \begin{bmatrix} \frac{\partial C_1}{\partial X_1} & \frac{\partial C_1}{\partial Y_1} & \frac{\partial C_1}{\partial \theta_1} & \cdots & \frac{\partial C_1}{\partial X_n} & \frac{\partial C_1}{\partial Y_n} & \frac{\partial C_1}{\partial \theta_n} \\ & \vdots & & \ddots & & \vdots & \\ \frac{\partial C_m}{\partial X_1} & \frac{\partial C_m}{\partial Y_1} & \frac{\partial C_m}{\partial \theta_1} & \cdots & \frac{\partial C_m}{\partial X_n} & \frac{\partial C_m}{\partial Y_n} & \frac{\partial C_m}{\partial \theta_n} \end{bmatrix}$$

Equation 4.4

The partial derivatives for the navigation capability functions were as follows:

$$\frac{\partial C_j}{\partial X_{Ri}} = \begin{cases} \frac{M\pi(X_{Ri} - X_{Tj})}{2Dk} * \sin\left(\frac{\pi}{2} * \frac{D}{k}\right)^{M-1} * \cos\left(\frac{\pi}{2} * \frac{D}{k}\right) * \frac{1}{10^{\sin\left(\frac{\theta_i - \alpha}{2}\right)^4}}, & D < k \\ \frac{-\pi k(X_{Ri} - X_{Tj})}{2D^3} * \sin\left(\frac{\pi}{2} * \frac{N}{N_{opt}}\right)^{M-1} * \cos\left(\frac{\pi}{2} * \frac{N}{N_{opt}}\right) * \frac{1}{10^{\sin\left(\frac{\theta_i - \alpha}{2}\right)^4}}, & D \geq k \end{cases}$$

Equation 4.5

$$\frac{\partial C_j}{\partial Y_{Ri}} = \begin{cases} \frac{M\pi(Y_{Ri} - Y_{Tj})}{2Dk} * \sin\left(\frac{\pi}{2} * \frac{D}{k}\right)^{M-1} * \cos\left(\frac{\pi}{2} * \frac{D}{k}\right) * \frac{1}{10^{\sin\left(\frac{\theta_i - \alpha}{2}\right)^4}}, & D < k \\ \frac{-\pi k(Y_{Ri} - Y_{Tj})}{2D^3} * \sin\left(\frac{\pi}{2} * \frac{N}{N_{opt}}\right)^{M-1} * \cos\left(\frac{\pi}{2} * \frac{N}{N_{opt}}\right) * \frac{1}{10^{\sin\left(\frac{\theta_i - \alpha}{2}\right)^4}}, & D \geq k \end{cases}$$

Equation 4.6

$$\frac{\partial C_j}{\partial \theta_{Ri}} = \begin{cases} -2 * \sin\left(\frac{\pi}{2} * \frac{D}{k}\right)^{M} * \cos\left(\frac{\theta_i - \alpha}{2}\right) * \sin\left(\frac{\theta_i - \alpha}{2}\right)^3 * \ln(10) \frac{1}{10^{\sin\left(\frac{\theta_i - \alpha}{2}\right)^4}}, & D < k \\ -2 * \sin\left(\frac{\pi}{2} * \frac{N}{N_{opt}}\right) * \cos\left(\frac{\theta_i - \alpha}{2}\right) * \sin\left(\frac{\theta_i - \alpha}{2}\right)^3 * \ln(10) \frac{1}{10^{\sin\left(\frac{\theta_i - \alpha}{2}\right)^4}}, & D \geq k \end{cases}$$

Equation 4.7

$$D = \sqrt{(T_x - R_x)^2 + (T_y - R_y)^2}$$

Equation 4.8

The partial derivatives represented how a change in either X_i, Y_i, or θ_i changed the overall capability C. Once calculated, the partial derivatives were inserted into the Jacobian matrix so that it may then calculate desired motion.

C. Primary Task Controller

Since the Jacobian matrix could be calculated from the partial derivatives of C, one could now relate unit motion to changes in capability. A change in capability $\dot{C}$ was determined by changes in X, Y, or θ multiplied by the Jacobian matrix. Therefore Equation 4.9 represents changes in C as a function of changes in X, Y, and θ.

$$\begin{bmatrix} \dot{C}_1 \\ \vdots \\ \dot{C}_M \end{bmatrix} = J * \begin{bmatrix} \dot{x}_1 \\ \dot{y}_1 \\ \dot{\theta}_1 \\ \vdots \\ \dot{x}_N \\ \dot{y}_N \\ \dot{\theta}_N \end{bmatrix}$$

Equation 4.9

Since the goal was to achieve a desired capability, C^d, the desired change in C was equal to the difference between the current capability and the desired capability, multiplied by a gain variable as seen in Equation 4.10.

$$\begin{bmatrix} \dot{C}_1 \\ \vdots \\ \dot{C}_M \end{bmatrix} = K_p \left(\begin{bmatrix} C_1^d \\ \vdots \\ C_M^d \end{bmatrix} - \begin{bmatrix} C_1 \\ \vdots \\ C_M \end{bmatrix} \right) = J * \begin{bmatrix} \dot{x}_1 \\ \dot{y}_1 \\ \dot{\theta}_1 \\ \vdots \\ \dot{x}_N \\ \dot{y}_N \\ \dot{\theta}_N \end{bmatrix}$$

Equation 4.10

Now Equation 4.10 was manipulated to find the desired changes in X_i, Y_i, and θ_i as a function of desired change in capability, $\dot{C}$. This allowed the controller to achieve its primary objective by providing directional commands to each individual unit until the desired capability was reached. In order to change Equation 4.10 into a function of capability change instead of X, Y, and θ, the pseudo inverse of the Jacobian matrix had to be taken. One was not able to simply take the inverse of the Jacobian because it was not necessarily a square matrix. Therefore, one had to take the pseudo-inverse to accommodate such uncertainty. The equation for the pseudo-inverse is as follows:

$$J^+ = J^T (J J^T)^{-1}$$

Equation 4.11

where J^+ is the pseudo-inverse and J^T is the transpose of the Jacobian matrix. One can now solve Equation 4.10 in terms of capability in order to calculate desired motion.

$$\dot{q} = \begin{bmatrix} \dot{x}_1 \\ \dot{y}_1 \\ \dot{\theta}_1 \\ \vdots \\ \dot{x}_N \\ \dot{y}_N \\ \dot{\theta}_N \end{bmatrix} = J^+ * K_p \left(\begin{bmatrix} C_1^d \\ \vdots \\ C_M^d \end{bmatrix} - \begin{bmatrix} C_1 \\ \vdots \\ C_M \end{bmatrix} \right)$$

Equation 4.12

Once one has $\dot{q}$, the various velocity commands could be provided to each unit so that they may approach the objective location in the most efficient way possible. Because this part of the project revolved around simulation, the controller did not actually send velocity commands to individual robots. Instead, the controller simply updated the simulated units' positions by multiplying the velocity command by a time constant and adding it to the Robot's previous position. By continuously updating the units' positions, one could see in Figure 23 how the robots were able to move in the shortest path possible towards the target location and orient themselves in the correct direction.

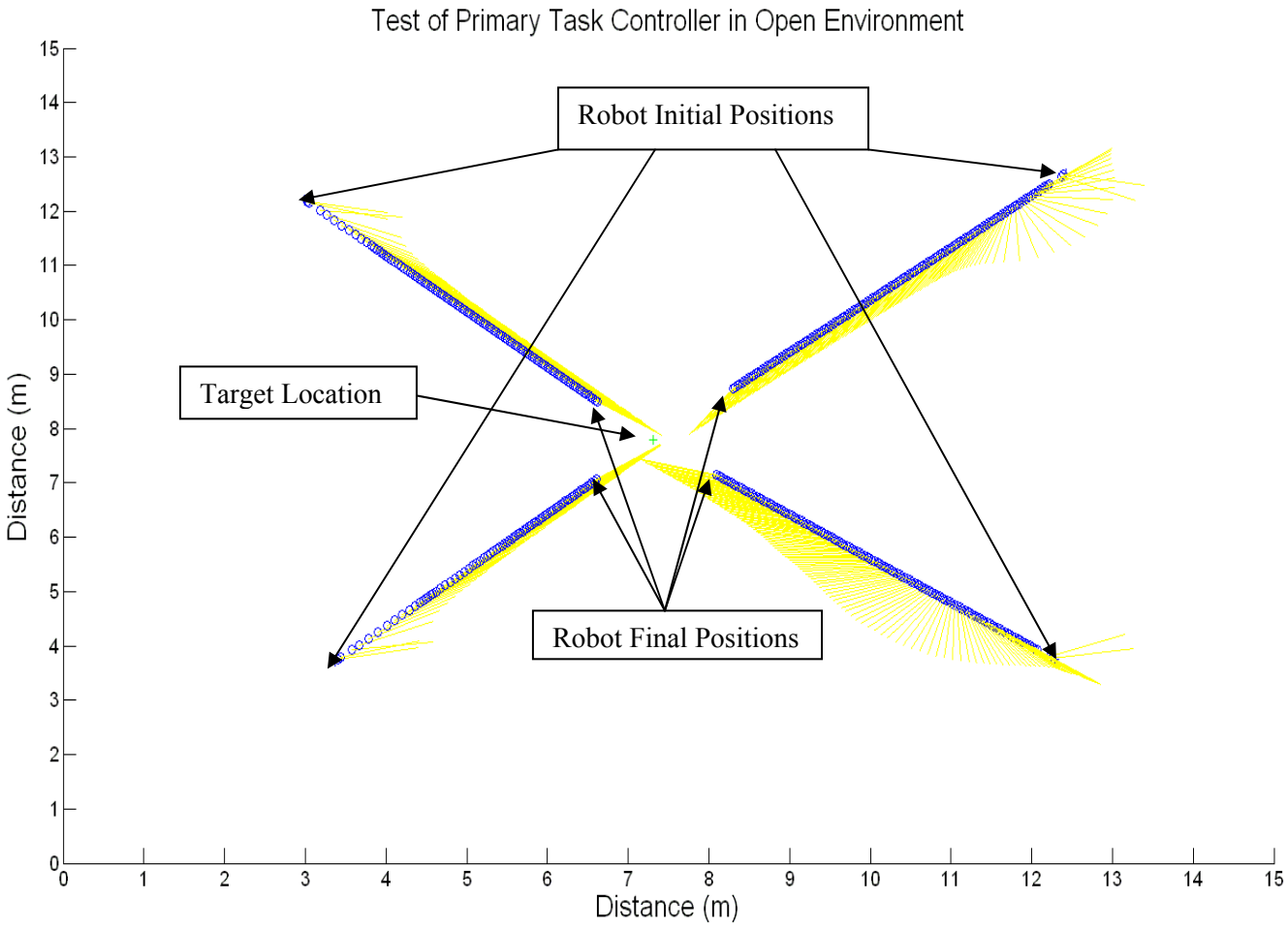

Figure 23: Visual simulation of 4 robots' movements over time.

In Figure 23, one could see the initial position and how the robot drove towards the target location using the Primary Task Controller. Each blue circle represented a robot position at a certain point in time. The yellow line represented the robot's current heading and it was "k" meters long. Therefore, in the final robot positions, one could easily see how each individual unit was oriented towards the target location and was approximately "k" distance away.

The simulation showed that the robots were able to drive towards the target location while continuously turning to orient themselves in the correct direction. It was important to note that throughout the course of this simulation, the units were assumed to be holonomic in nature. Locomotive constraints were taken into account in the second half of the project. Another important note was that the robots took the shortest path possible to reach the desired location. This was a result of the pseudo-inverse which returned the local minimum velocity solution in order to achieve the desired capability. This was critical to the efficiency of the swarm. At this point in the project the controller development had successfully completed the primary objective for the swarm. However, as previously discussed, achieving the secondary objectives were critical to the success of the swarm in unknown environments and make it an efficient choice for reconnaissance based operations.

D. Resource Allocation and Efficiency

Another capability of this controller was that it was able to account for multiple objectives. Since a swarm has at least several units in it, it should be able to divide their capabilities to accomplish multiple tasks. Therefore, one of the original objectives for this controller was to automatically allocate the swarms resources in order to accomplish multiple

tasks at the same time. For reconnaissance-based operations, this usually entailed gathering information on multiple locations simultaneously.

While this was considered a secondary objective, it was actually controlled by the primary controller. When developing the Jacobian matrix for the swarm it was possible to include multiple capability functions as well. As seen in Equation 4.4, the Jacobian matrix was able to handle anywhere from 1 to *n* individual robotic units, as well as 1 to *m* target locations. As the pseudo-inverse was calculated and the task variable equations were established (Equation 4.12), the velocity commands were able to accommodate different capabilities to different locations. This assumed that the sum of all desired capabilities for every target location does not exceed the potential capability for the swarm.

The primary controller was also responsible for the overall efficiency of the swarm. This was one of the fundamental principles behind the pseudo inverse of the Jacobian matrix. In this case it provided the local minimum velocity commands required to converge to the desired capability. This allowed the swarm to accomplish the mission while attempting to use the least amount of resources required.

5. Secondary Controller

A. Secondary Objectives

Once able to establish that the primary controller was successfully able to deliver the appropriate amount of capability to the target location, it was necessary to start taking into consideration the secondary objectives that would not only allow the swarm to operate in any unknown environment, but also allow the swarm to be a viable, efficient choice for

reconnaissance based operations. Without these secondary objectives, there would be chaos as the members of the swarm would constantly collide into obstacles, each other and the target location itself. The swarm would, in essence, often be rendered useless and would fail to be a practical choice. It was felt that in order for the swarm to successfully navigate and unknown environment and meet a set of necessary requirements the following secondary objectives must be met.

- Avoid collisions with obstacles
- Avoid collision between individual robots
- Avoid collision with objective location
- Achieve Line of Sight Capability
- Minimize the effects of local minima
- Automatic resource allocation
- Efficient use of power
- Low response time for emerging needs
  Capable of flexibility in specification of mission and objectives

These objectives represent the minimum ability needed to prevent the swarm from causing significant damage to itself or the operational environment.

One way in which some of these secondary objectives could be encoded was by the use of artificial potential fields (APFs).[10] Using APFs, each individual unit could be "attracted" to optimal locations while being "repelled" by obstacles, other robots, or undesired areas. The sum of all of these vectors could then be added into one resultant vector which controlled the unit's overall velocity and motion. The further a unit was from a desired position, the greater the attractive force was, causing a higher velocity at greater distances. However, as the unit approached the location the magnitude of the force decreased until it eventually reached zero

when the unit was directly over the position. The repellant force on the other hand did the exact opposite, causing the force magnitude to increase once the robot approached the obstacle or undesired location. By adding these two forces, each individual unit automatically prioritized secondary objectives based on location and motion. One thing to note was that the repulsive force from the APF was nonlinear and would exponentially approach infinity as the distance to impact neared zero. Equation 5.1 to Equation 5.3 show how an APF is calculated for obstacle avoidance. It was assumed that every robot was equipped with some sort of range sensor so that it was able to calculate the distance and bearing to the closest obstacle

$$D_{Obst} = \sqrt{(X_{Ri} - X_O)^2 + (Y_{Ri} - Y_O)^2}$$

Equation 5.1

$$APF_{RepelX} = K_{APF1} * \cos\left(\frac{K_{APF2}}{D_{Obst}}\right)$$

Equation 5.2

$$APF_{RepelY} = K_{APF1} * \sin\left(\frac{K_{APF2}}{D_{Obst}}\right)$$

Equation 5.3

Since the swarm was intended to operate in an unknown environment, it was necessary for the robots to be able to react to their surroundings without having to be specifically instructed on how to do so. This was the foundation for behavior-based control theory.[11] Using a behavior-based controller to achieve the secondary objectives provided the required flexibility to operate in an unknown environment without losing the guaranteed success of the systems-theoretic primary controller.

B. Avoiding Obstacles

Perhaps the most important ability of the individual units in a swarm of robots was to avoid obstacles. This could be done in variety of different ways however the concept of artificial potential fields (APFs) were used as previously discussed. This allowed for a more smooth approach and avoidance of the obstacles. The first step to avoiding the obstacle was finding the closest point on the obstacle to the robot. When the user creates the environment, the obstacles were constructed by selecting an array of vertices and connecting the selected points. Therefore, depending on the number of vertices, it was possible to have obstacles that were very rigid in nature or ones that have more rounded edges. The first approach taken to solve this problem involved first finding the closest obstacle. This was accomplished by simply searching through the entire array of vertices for the closest one and assigning the obstacle as the obstacle to be avoided. A few issues were encountered with this however. First, it was possible for the obstacles to have very long flat sides. Based upon the current search technique a robot could be positioned very close to a flat side of obstacle A; however, the robot may think that obstacle B is closer because obstacle A's vertices are more distant. This problem is evident in the situation shown in Figure 24.

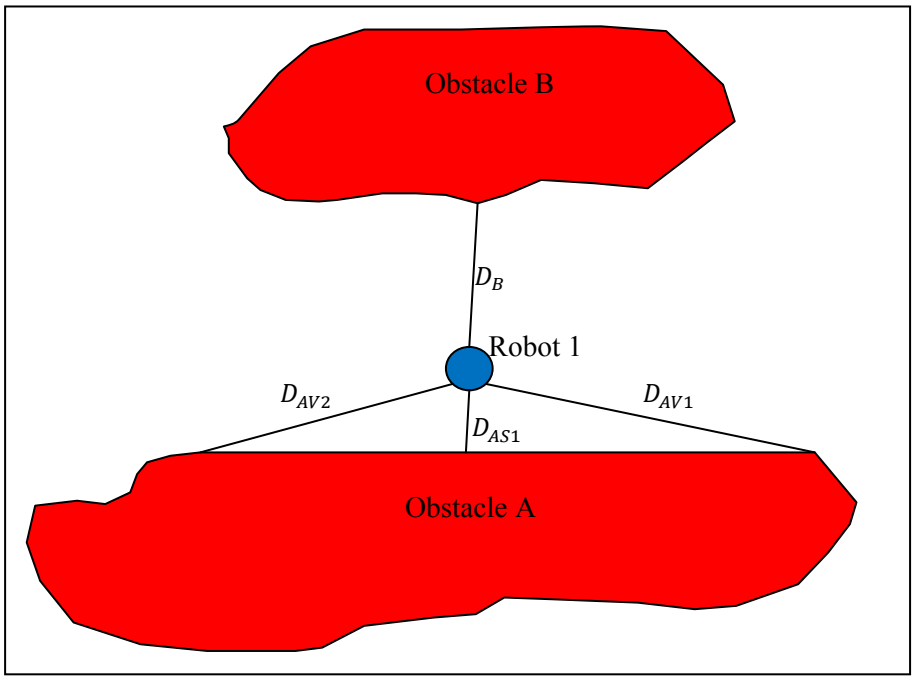

Figure 24: Visual representation of issues with initial search technique.

As one can see in Figure 24, D_{AS1} was obviously the shortest distance; however, by only searching through obstacle vertices, the controller actually thought that obstacle B was closer because D_B was shorter than D_{AV1} and D_{AV2}. Therefore 2 major changes were implemented. First, instead of finding the closest obstacle, the controller found the closest vertex for every obstacle. Second, once the closest vertex was found, an intersection function was run to determine if the closest distance to the obstacle was actually one of its sides as evident with Obstacle A in Figure 24. This was accomplished by determining if a line from the robot to the obstacle's side ever intersected the side at a perpendicular angle. With the search program complete, each unit now knew the closest distance to every visible obstacle in the environment

The next step was to determine if any of the obstacles posed a threat to the robot. This was accomplished by checking all of the closest distance to obstacles to see if any were less than my determined "Safe Zone". If an obstacle was determined to be within the "Safe Zone", a repulsive APF was created to drive the robot in the opposite direction. Figure 25 shows how

when a robot approaches an obstacle, the repulsive force grows, thereby having a greater affect on the resultant vector.

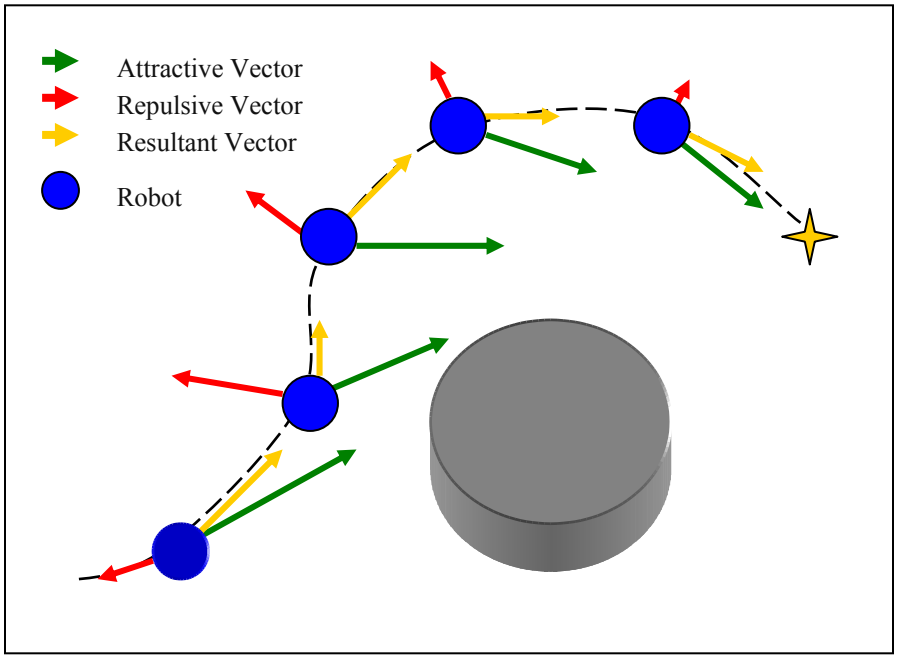

Figure 25: Basic obstacle avoidance using APFs.

All of the repulsive APFs for the every obstacle within the safe zone were then summed together and resulted in a resultant APF which would direct the robot in a path to avoid all obstacles. This was done for every robot in the swarm.

C. Avoiding Other Robots and Objective Location

This was very similar in principle to avoiding obstacles. The first thing was to determine the distance to every other robot in the swarm. If any of these robots were in the same "Safe Zone" a repulsive potential field was generated in the opposite direction and its magnitude depended on the actual distance between the robots. The interesting fact for this case was that every single robot was constantly searching for the distance to the other robots. So whenever

one was required to repel from another, that force actually acted on both robots. This provided higher effectiveness because the robots were constantly avoiding each other through the most efficient means.

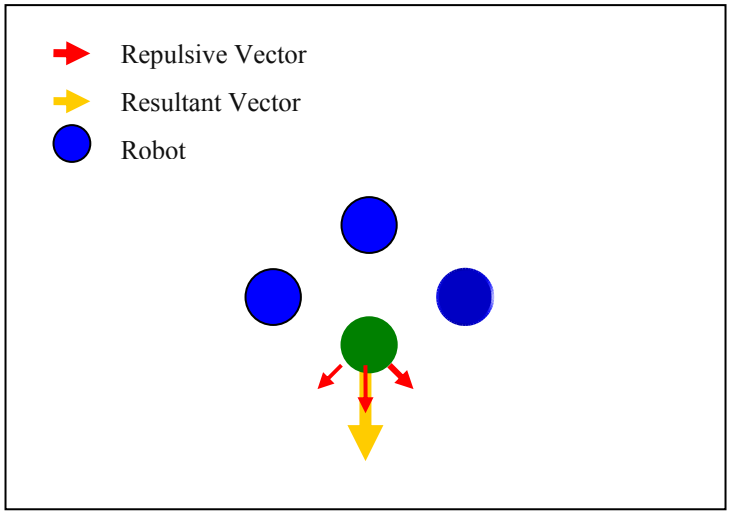

Figure 26: Repulsive APFs from 3 other robots acting on one.

Figure 26 shows the basic concept of how the repulsive APF from another robot is combined with the rest in the swarm to produce a resultant vector.

It was also very important to avoid the objective location. In many cases the actual location could be something sensitive in nature or highly visible. Another example would be if the swarm wanted to check a specific location for a landmine or IED. One does not want the robots to drive over the landmines; they should only approach the location and return intelligence. This was accomplished the same way as avoiding the other robots.

D. Achieve Line of Sight Capability

It is common sense that a camera system cannot deliver any capability to a certain location if it is not able to see the target. Therefore, it becomes imperative for every unit to be

aware of its line of sight capability. There were a variety of ways for approaching this issue. The first step was to calculate a relative bearing to the target location from the robot's current position. A search algorithm was created that would return the max and min angles to every obstacle in the environment. If the bearing to the target was in between the max and min angles for a certain obstacle the controller investigated even further. The next step was to check the distance to the target and to see if it was in front of or behind the obstacle.

Once it was determined that there was no line of sight capability, the next step was to direct the robot around the obstacle in the most efficient way possible. The controller had to first decide which direction to take around the object, as there were two possible solutions. Using the same search algorithm used earlier, the robot was able to calculate the max and min angles for the obstacle blocking its view. Then, based upon the difference between those angles and the bearing to the target, the robot moved in the direction with the smallest difference.

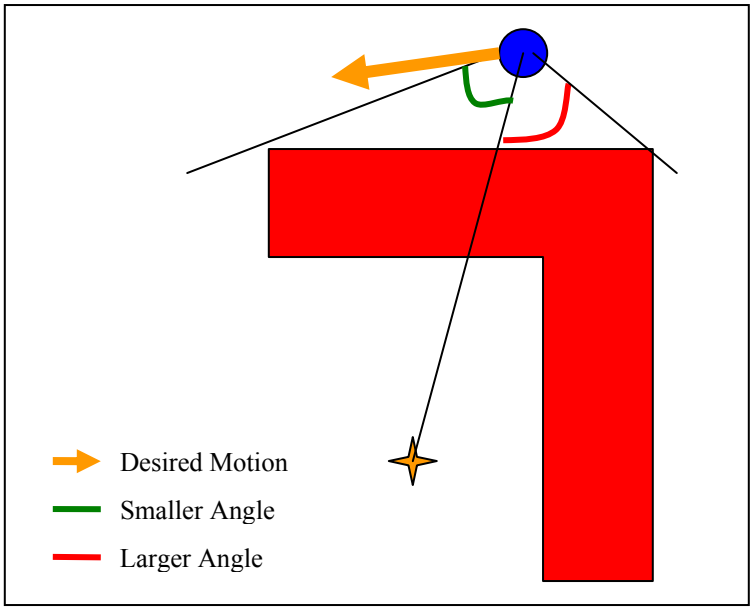

Figure 27: Single Robot achieving line of sight.

There were some possible concerns that arose however. Due to the uncertainty of the environment, it was impossible for the robot to be aware of what it could not actually see. While in simulation the controller could be modified to calculate the most efficient way, it was necessary to make the decision based upon the information provided by the robot. This was because in a real world scenario, the information from the robot is all the information the controller has to make a decision. Therefore, it was possible the robot may unknowingly choose the least efficient direction; however, once the original decision was made for which direction to pursue, the robot must stick with it to prevent oscillation.

E. Minimize the effects of local minima

When dealing with artificial potential fields, a concept known as local minima is often encountered. The local minima is a point at which the sum of all the potential fields is equal to zero. Basically, when combined, all of the forces cancel out and the robot is permanently stuck in one location. There were ways to recognize this event and the controller had to be able to direct the robot out of this position. The solution used in my controller involved the archiving of the previous robot positions. Throughout the course of the simulation, the previous ten positions for each robot were stored in a data array. From this data, one was able to calculate the mean position of the robot over the course of the past ten time intervals. If the robot's current position was within a certain range of the mean position of this history vector, then the controller generated a repulsive force that was applied to the robot. This force was designed to be great enough to propel the robot out of the local minima position. While the effects of local minima cannot be completely eradicated with only the information from the robots, it was possible to

minimize the impact. Figure 28 shows one particular solution for how this controller handles the local minima issue.

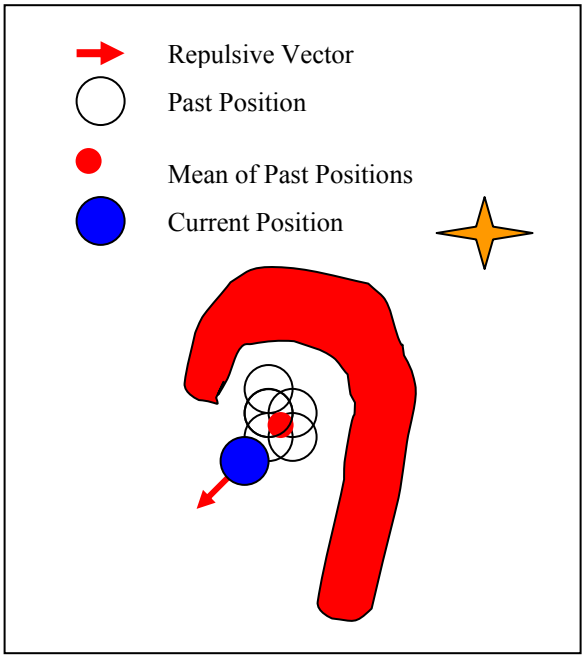

Figure 28: Representation of how to minimize the effects of local minima

F. Mission Flexibility

While the specific focus of this project was reconnaissance-based operations, it was a goal that this controller also be applicable to a variety of different needs in the future. Therefore, it was necessary to maintain a certain level of flexibility throughout the development process. One of the major things to avoid was putting too much reliance on information provided by means other than the robots themselves. In this first semester simulation, the simulation included every aspect of the environment because it was user defined. However, one has to be cautious that he or she did not provide control directions or other commands to the robots that were based upon information not procured by the robots themselves. Because the swarm was designed to

operate in an unknown environment, the only information that can be used for control was that which was provided by the robots' sensors.

In order to use this controller in a different operation, one needs to only establish what the specific capability being delivered is and find a way to quantify it. For example, if the goal of the swarm was to provide light on a certain location, one could measure the intensity of the light at the location as the delivered capability. Then, based upon a unit's ability to provide light from a certain position, one can calculate the total capability being delivered and instruct the robots to move accordingly.

6. Final Swarm Controller

A. The Null Space

Once the primary and secondary controllers were completed, one must include them into one master controller for the swarm. Because the primary task was to deliver the desired capability on target, one must find a way to have the controller also be aware of the secondary tasks without affecting the primary objective. This was accomplished by projecting the secondary tasks onto the null space of the controller. The null space of the main controller is the set of motions of the swarm at any given point that would have no impact on the primary capability. The null space is a function of the swarm state, the environment and the capability. When properly defined, the robots were provided with the freedom to move about and have no effect on the primary controller. There is an easy way to think about the null space of a controller. If two men were told to carry a couch through a room, the primary controller would provide them with the desired direction and speed. However, they are not told how to move

across the room. They could orient the couch in many poses, even walk in small circles as they moved, and still accomplish the task of getting the couch to its target location. This was the same for the swarm controller. The robots were told to move in coordination to achieve capability on target points, but there were an infinite number of ways for them to move to achieve the goal. The secondary controllers, which were projected onto the null space of the primary controller, tell the robots to avoid obstacles, each other, and accomplish all of the other secondary tasks.

B. Final Controller

At this moment, the controller had two components: the pseudo-inverse of the control Jacobian to direct the swarm to its primary objective, and a sum of secondary controllers, encoded as different potential fields generated by the swarms interaction with its surrounding environment and itself. These had to be combined into one by projecting the potential fields onto the null space of the primary controller. Equation 6.1 represents the combined swarm controller.

$$\dot{q} = J^{+} * K_p \left(\begin{bmatrix} C_1^d \\ \vdots \\ C_M^d \end{bmatrix} - \begin{bmatrix} C_1 \\ \vdots \\ C_M \end{bmatrix} \right) + \boldsymbol{K_t} * (\boldsymbol{I} - \boldsymbol{J^{+}J}) * \boldsymbol{v}$$

Equation 6.1

As one can see it was simply the primary controller with an added operation on the end. This mathematical operation was the projection of ‘v’ onto the null space. The variable ‘v’ was a vectorized sum of all of the unit level controls. However, this does not include the heading

component, only the X and Y parts. It does include unit level control for obstacle avoidance, robot avoidance, achieving line of sight and others. In order to project this onto the controller one must multiply it by the matrix $\boldsymbol{K_t} * (\boldsymbol{I} - \boldsymbol{J^+ J})$, where the pseudo-inverse of the Jacobian was multiplied by the Jacobian and subtracted from the identity matrix 'I'. K_t was a system gain value that was used to control the amount of influence the task space controller has on the entire swarm. It was very important to establish an appropriate balance between K_p and K_t because that determined how much the swarm was controlled by the primary or secondary controllers. If K_p was too high, the secondary controller was overwhelmed and the potential fields generated will not be enough to prevent the robots from colliding into obstacles and other things. In the end, the combination of these two controllers provided the swarm with the velocity vector $\dot{q}$, which provided individual velocity commands to the units in order to accomplish the primary and all secondary objectives.

7. Simulation Results

A. Selected Test Beds

In order to test the controller in a variety of different environments, five test beds were developed to perform the simulation. Each of these five tests beds was unique in some major way. This was important because the success of the swarm in different settings was critical to its overall purpose. Therefore, each test bed must assess one of the major concerns for the swarm and its environment. The selected test beds were as follows:

1. The robots start at opposite corners of the environment with no initial line of sight capability.
2. The robots approach the target from the same heading but have to navigate a field of 6 different obstacles.
3. The robots are somewhat spread out but approach a target that is completely surrounded by an obstacle.
4. The robots each possess unique lens systems as well as different locomotive constraints.
5. The robots converge on multiple target locations.

As one can see, each test bed has a major difference that makes it unique to the others. Success of the swarm can only be achieved by successfully navigating and delivering appropriate capability in every one of the test beds.

B. Test Bed #1

As previously mentioned this test bed consisted of 4 robots all approaching a central target location from opposite corners of the environment. This was designed to test the robots' abilities to accomplish an objective while experiencing separation from the swarm itself. It also tested each unit's ability to navigate and avoid obstacles from a variety of directions and starting positions. This was the simplest of all the test beds. This target point has a desired capability of 3.95. The reason for this number was to ensure that each unit reaches as close as possible to its maximum capability, but it will save time by not waiting for the final small movements to

reach absolute maximum. Each of the robots was equipped with a 35 mm camera lens and they all have the same kinematic constraints.

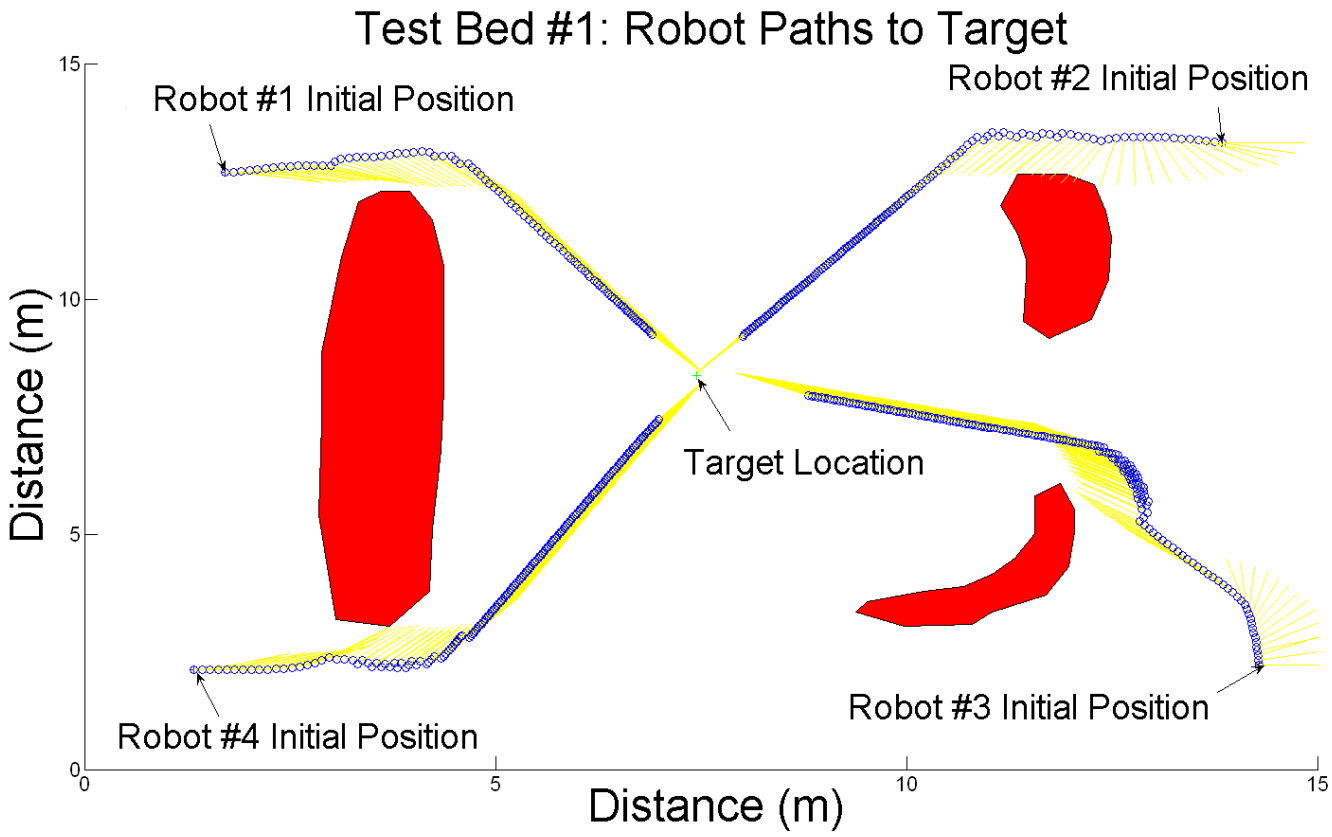

Figure 29: Test Bed #1- The paths of the robots over time.

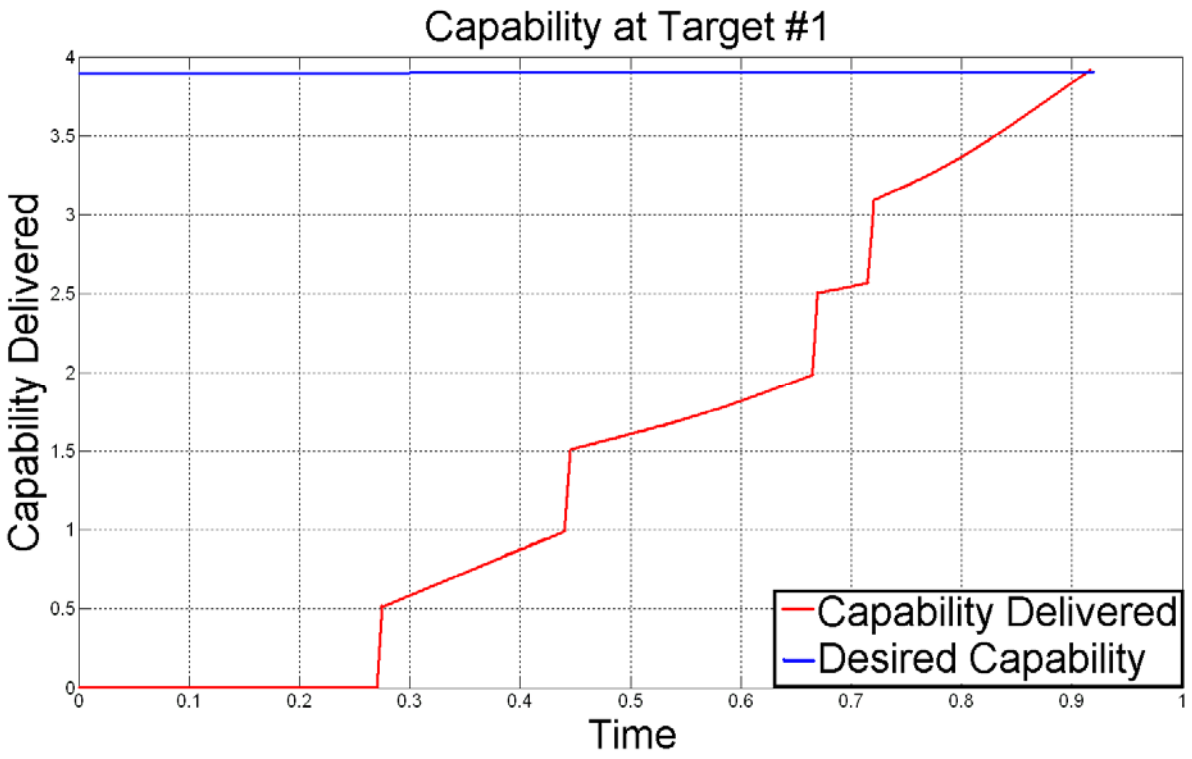

Figure 30: Test Bed #1 - The Capability over time.

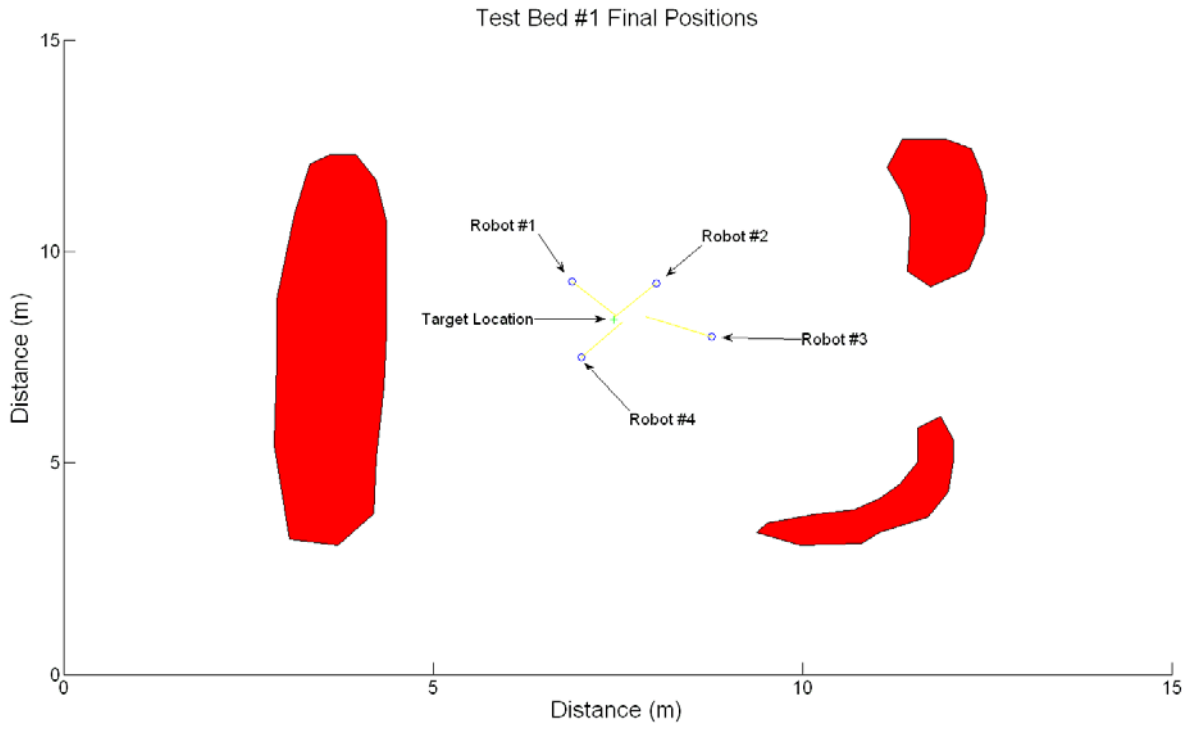

Figure 31: Test Bed #1 - Final position and orientation of the robots.

As one can see in Figure 29, the robots were successfully able to determine the shortest path around the obstacles and achieve line of sight capability. From there they converged on the target location and positioned themselves in a manner to deliver the desired capability. While avoiding the obstacles, the robots maintained the safe distance of 1 meter. This distance was encoded into the controller as the environments "safe zone".

In Figure 30, one can easily see at what time the robots achieved line of sight. There are four major spikes in delivered capability and these spikes correlate with the four robots in the swarm. When the target all of a sudden appeared from behind an obstacle or entered the robots field of view, there was a sudden increase in capability delivered. The smooth, gentle slopes correspond to the robots gradually approaching the target and increasing their capability. Once, the swarm reached the desired capability, the controller stopped and the robots were left in their final position as seen in Figure 31. The controller, which used the smooth navigation capability functions, was unaffected by the discontinuities in the delivered capability and was able to achieve the objectives.

C. Test Bed #2

The purpose of this test bed was to investigate the swarm's ability to not only avoid obstacles but other robots as well. As the swarm approaches a target point from the same heading, the units are often sent on collision courses with obstacles and each other. There are also locations where the units encounter a bottleneck and they must be able to navigate through it without any collisions whatsoever. It was how the swarm handles situations like this that determine its applicability in rough or complex environments. By successfully navigating this

setting the swarm proved that it was able to accomplish the basic functions of operational safety and endurance. The swarm and capability desired in this scenario was the same as test bed #1.

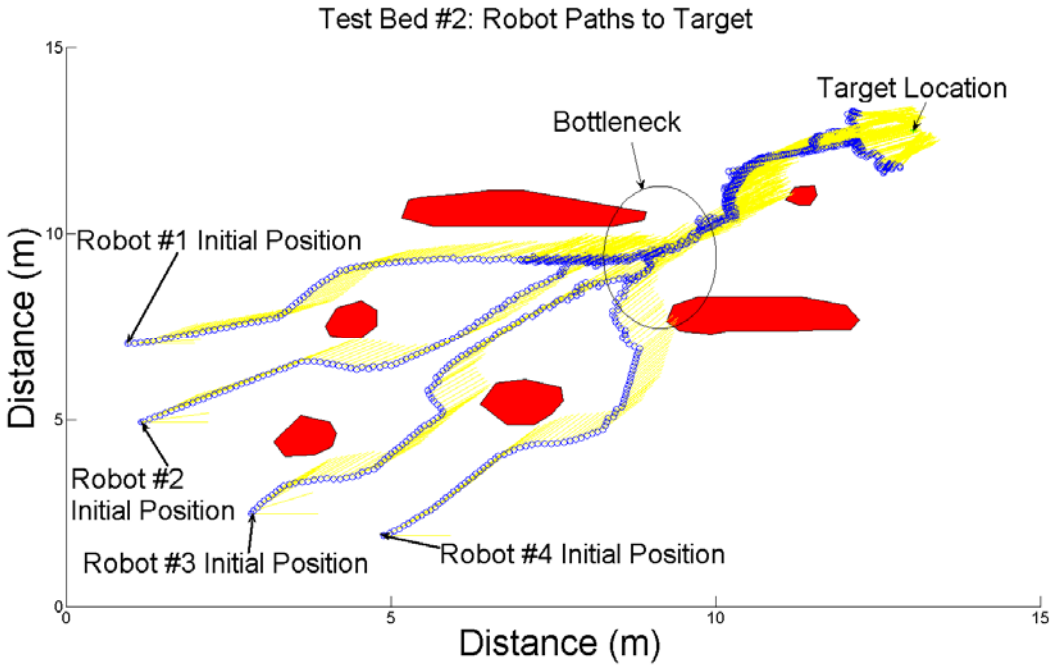

Figure 32: Test Bed #2- The robot paths over time.

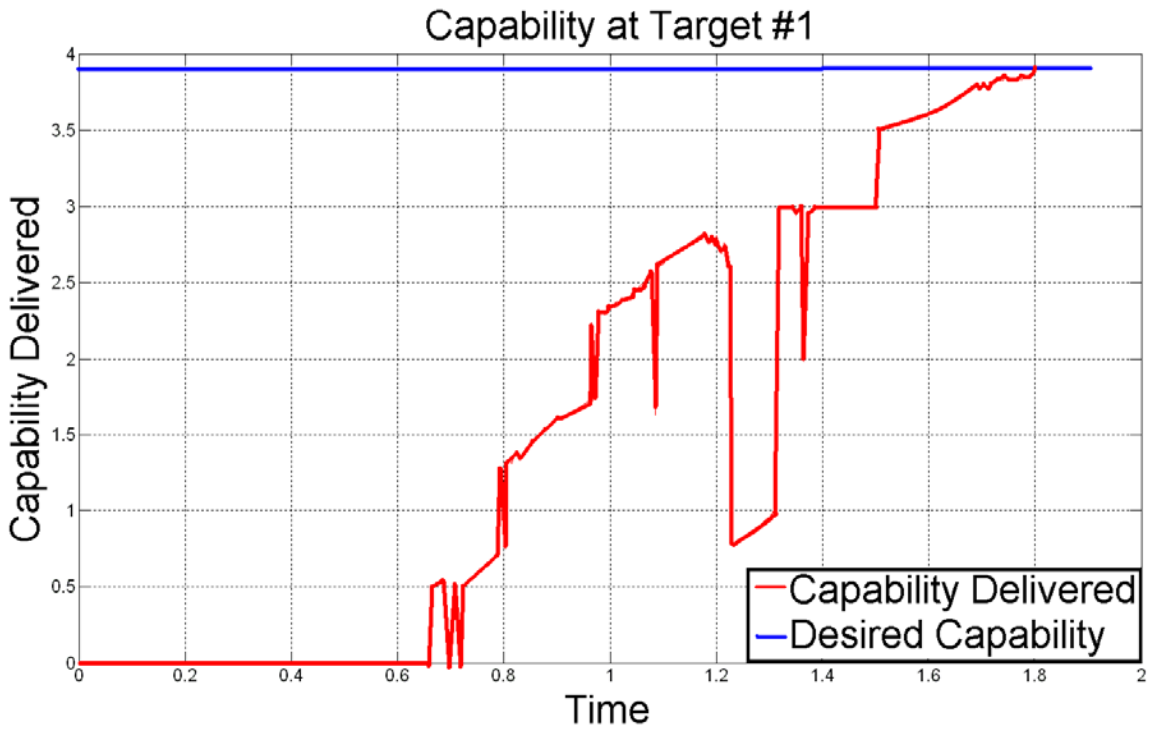

Figure 33: Test Bed #2 - The capability delivered over time.

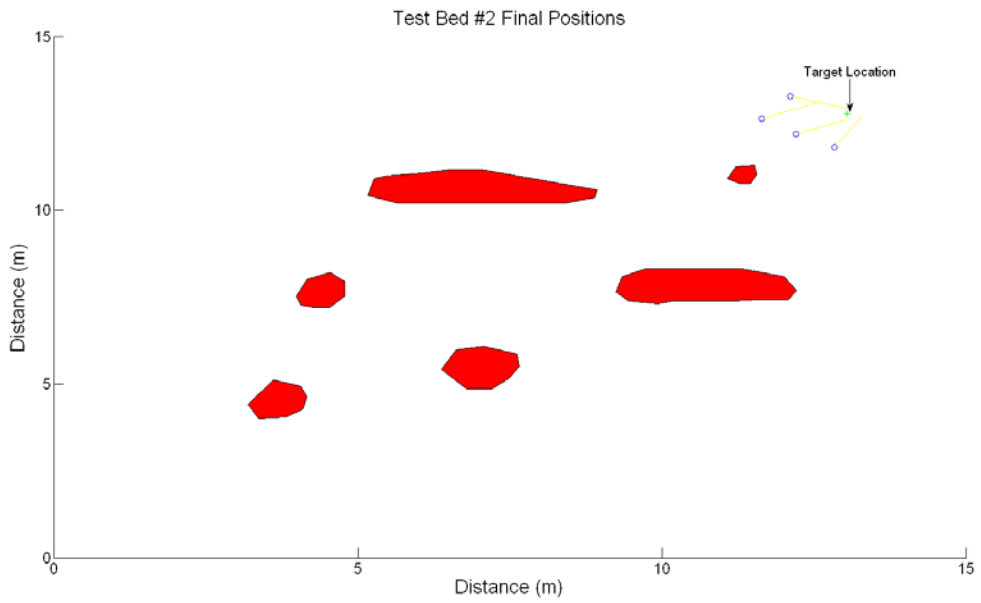

Figure 34: Test Bed #2 - Final position and orientation.

As shown, the robots were successfully able to avoid all the obstacles in this complex environment. Although the bottleneck in Figure 32 is deceiving, the robots proceeded through it in a single file manner. There were no collisions at any time, the paths just overlapped. One interesting thing about the final approach to the target was how the robots spread out at the last second. This was a beneficial side effect of the controller in the sense that the robots are attracted to the target by the primary controller but the task space controller prevents them from running into each other. Since they all approached from the same direction, it was necessary for the robots in the front of the line to shift to the side to allow the rest of the robots to approach. So as one robot approached, the repulsive APF generated between the two robots caused them to surround the target, thus providing multiple viewpoints and the ability to gather more information.

The sharp spikes in Figure 33 are a result of the target constantly disappearing behind obstacles and then reappearing as the robots maneuvered through the environment. Just like test

bed #1, as the robots approached the final location, there was a more smooth gradual increase in capability until the desired level was finally met. The swarm was very successful in this environment.

D. Test Bed #3

By searching for an object completely surrounded by an obstacle, the controller's ability to direct the robots in order to obtain a line of sight capability was put to the test. This was a major component of the task space controller. If the swarm was unable to achieve line of sight to the target, then the swarm was rendered useless. Therefore this ability becomes critical to mission success. In this environment, the robots started with moderate separation between them. As they approach the target, they decided on which way to proceed around the obstacle until the target was in line of sight. Then, depending on the amount of space in the obstacle, the robots were either able to surround the target location or had to remain in the opening of the obstacle.

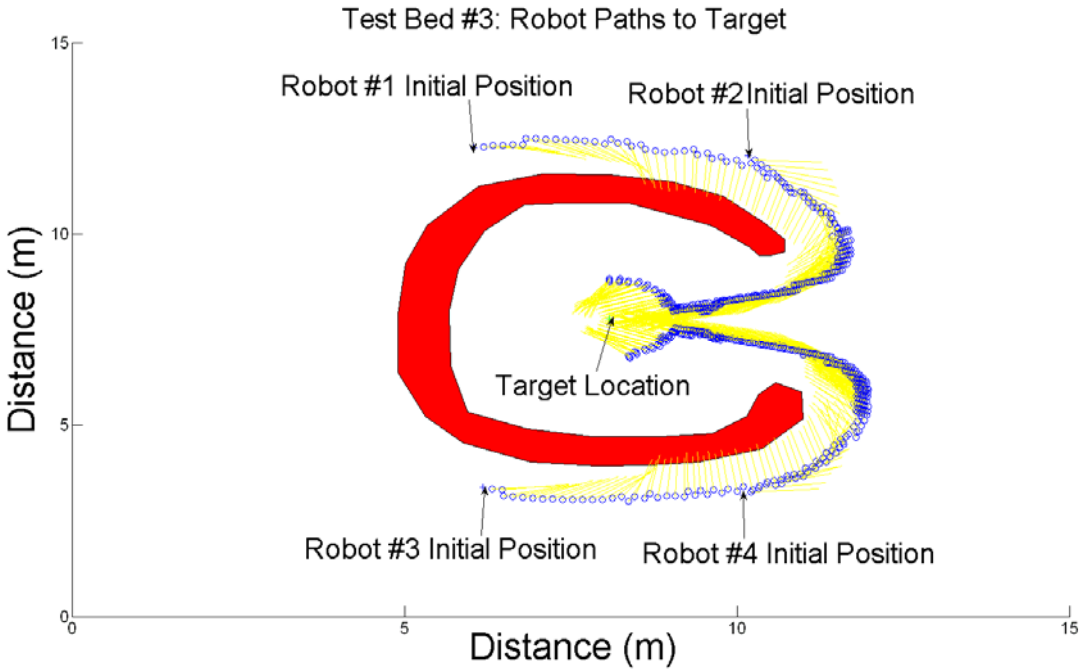

Figure 35: Test Bed #3 – Robot paths over time.

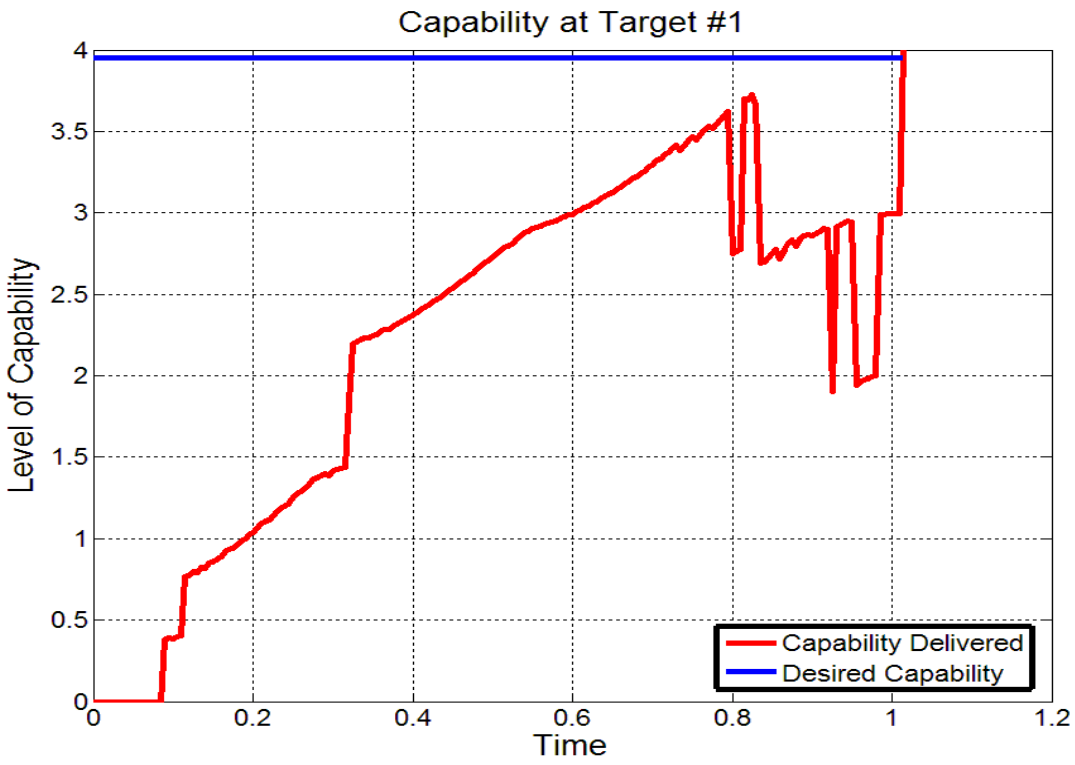

Figure 36: Test Bed #3 – Capability delivered over time.

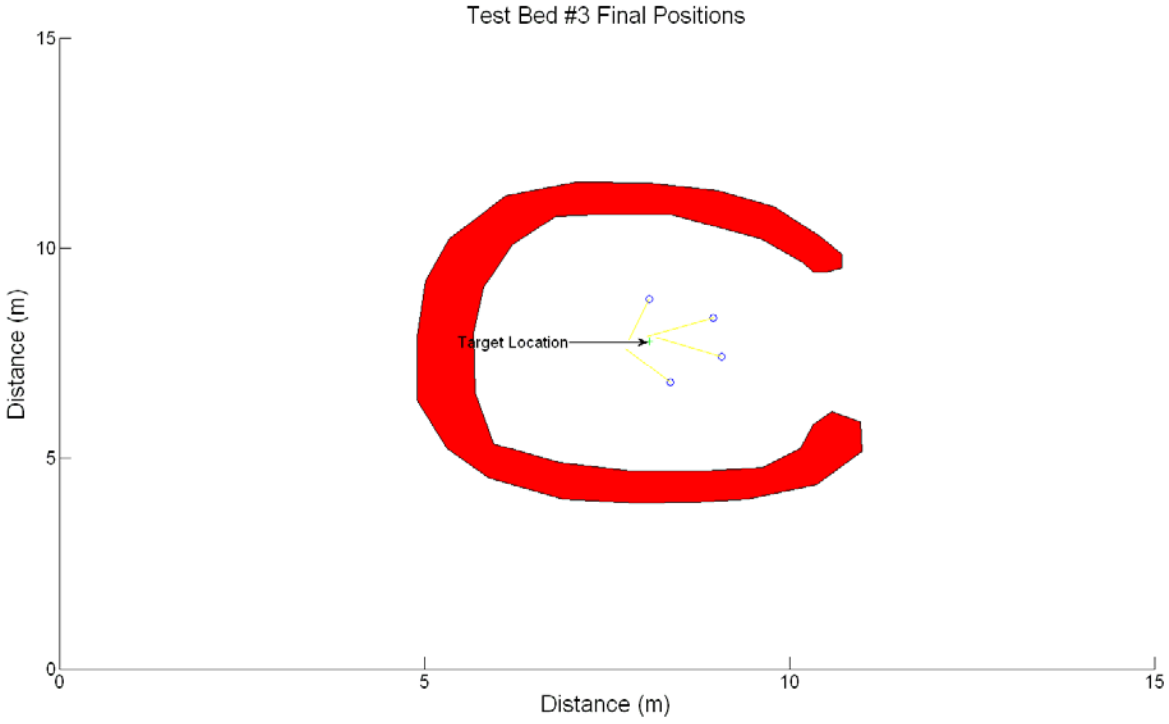

Figure 37: Test Bed #3 – Final position and orientation.

In this particular case, the controller worked very well. However, as will be discussed later, there were situations in which the robots became confused. This was due to a potential problem with the mathematics where a discontinuity occurs. All other aspects of the controller consistently worked however. In this test one can easily see how the robots maintained their distance from the obstacle until they located an opening through which they could approach the target. The sudden spike of capability at the end was a result of the robot turning last second and the target entering into the field of view.

E. Test Bed #4

Another major characteristic of the swarm being used by this controller was that it could be heterogeneous. These differences could be either the specific capabilities of each of the units or the locomotive constraints on the units themselves. In this test, there were both. Each lens system on the different robots had a different focal length. Therefore, as seen in Figure 6, there was a different optimal range for each focal length, with a 35 mm lens at about 1 m and the 150 mm lens at about 4.5 m. In the following simulation, the yellow line attached to each of the robots provided two valuable pieces of information. The first was the actual heading of the vision system. This was important because it actually shows which direction the robot is facing and was therefore crucial to determining field of view. The second was the length of the yellow line. As one can see in this simulation, the yellow lines for each unit were all of different lengths. The length of the yellow line corresponded to the distance *k* for that particular focal length. Therefore, when the tip of the line is right on the target location, it was easy to see that the robot was delivering maximum capability. The second difference between the units was the

max speed of each unit. Since some of the robots were able to move faster than others, they reach the target first and orient themselves on target. These two differences are critical to the controller being able to handle a heterogeneous swarm.

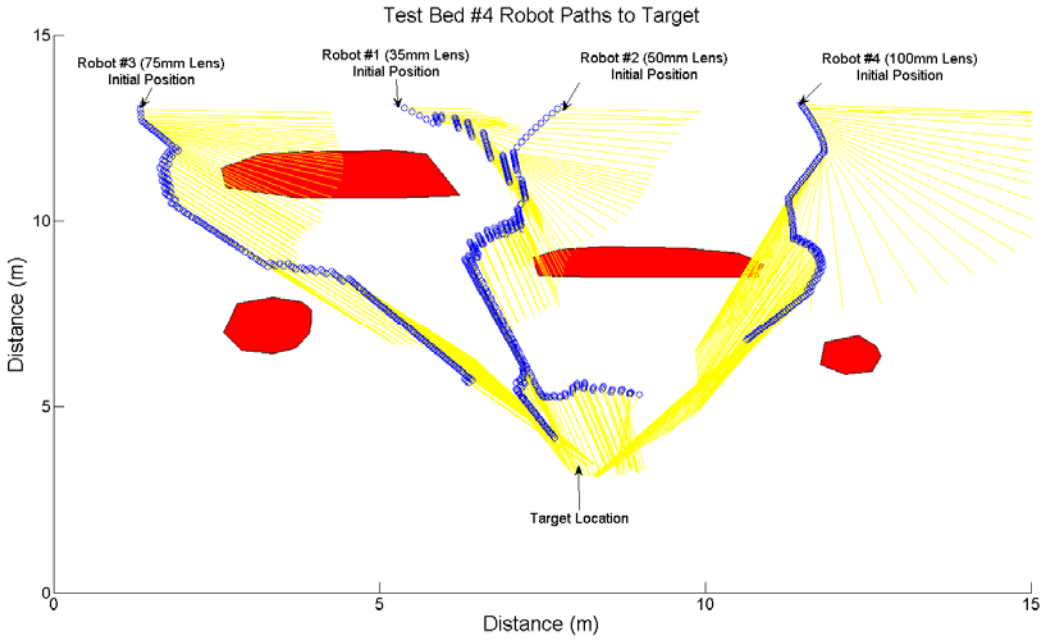

Figure 38: Test Bed #4 – Robot paths over time.

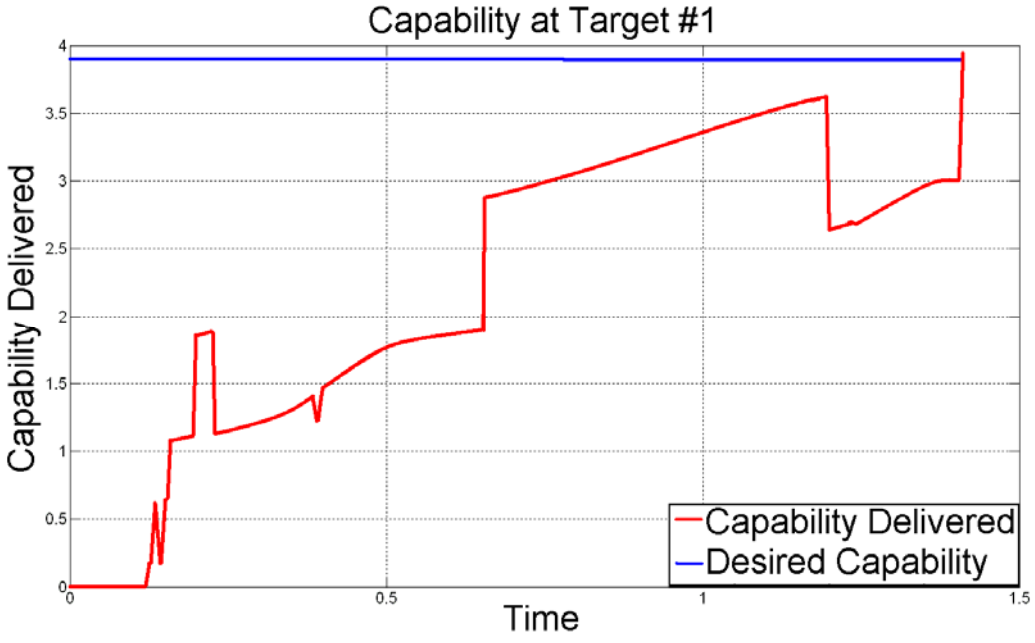

Figure 39: Test Bed #4 – Capability delivered over time.

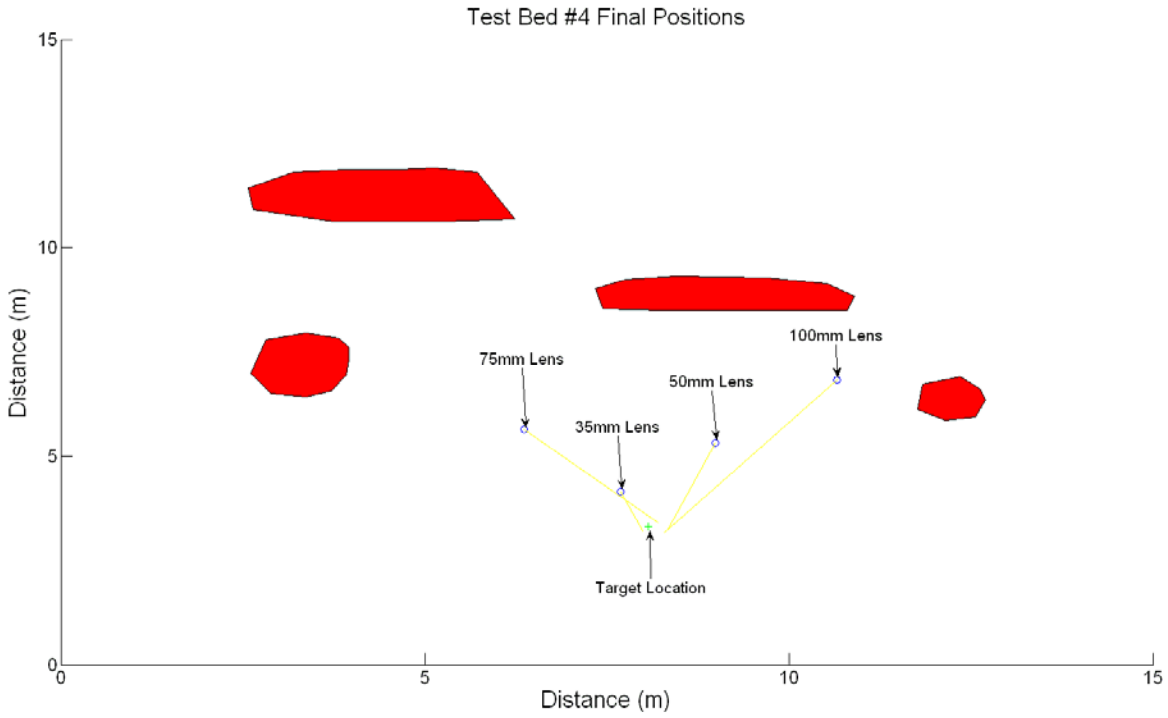

Figure 40: Test Bed #4 - Final position and orientation.

The controller was clearly able to handle the different vision systems with no difficulties whatsoever. As shown in Figure 40, each individual robots final position corresponded directly to its specific sensor capability. The robots equipped with longer focal length lens systems were located further away from the target.

F. Test Bed #5

The final test scenario assessed the ability of the swarm to allocate resources for multiple targets. As the swarm approached the two targets, they automatically are able to position the individual robots in a way that provided the desired capability on both targets. This scenario took a significantly longer time to achieve due to the discontinuity in the deliverable capability.

If the controller was based solely on the navigational capability then it would have a smoother approach and more efficient response. However, because the deliverable capability can drop from full capability to zero by simply rotating one degree, one can see a more rigid motion of the robots. There were many spikes and falls in the amount of capability delivered on the targets because the targets were constantly going in and out of the field of view.

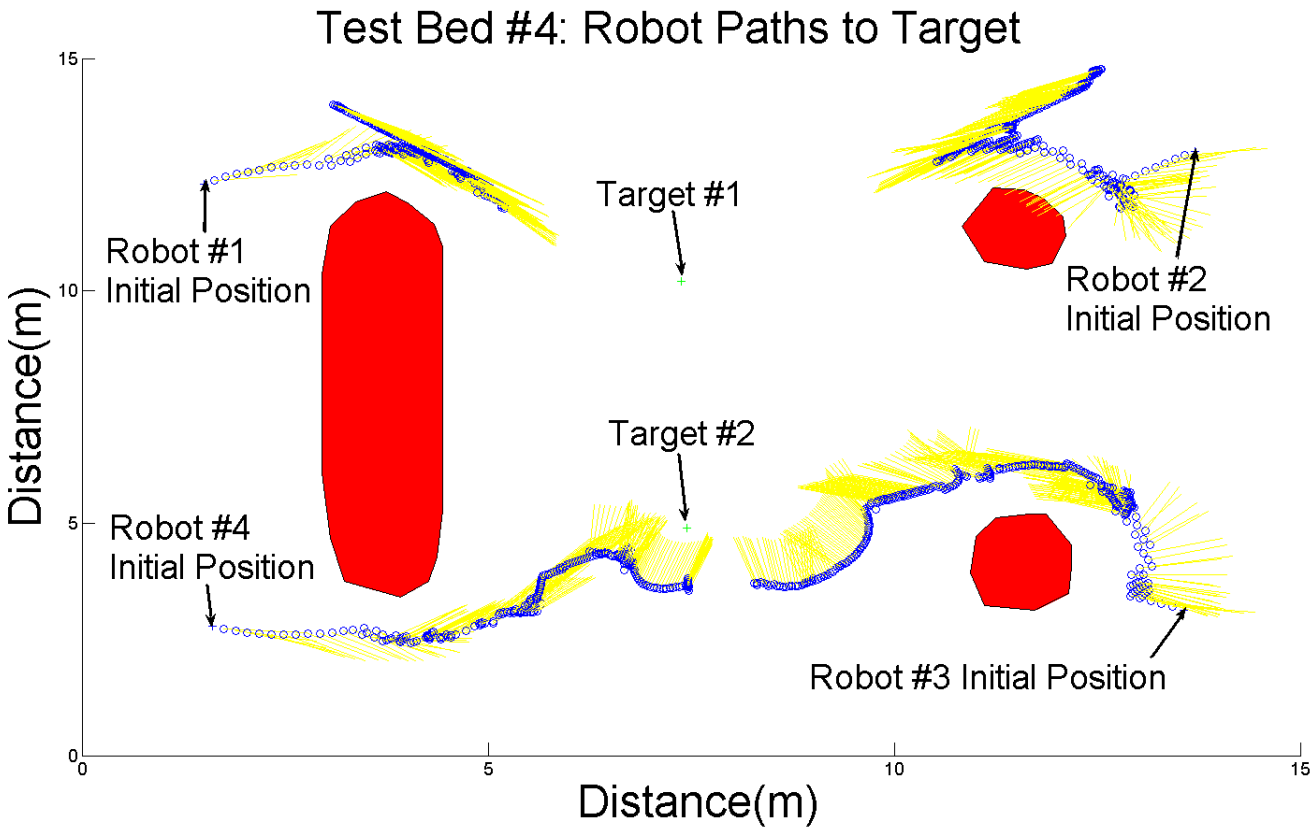

Figure 41: Test Bed #5 – Robot paths over time.

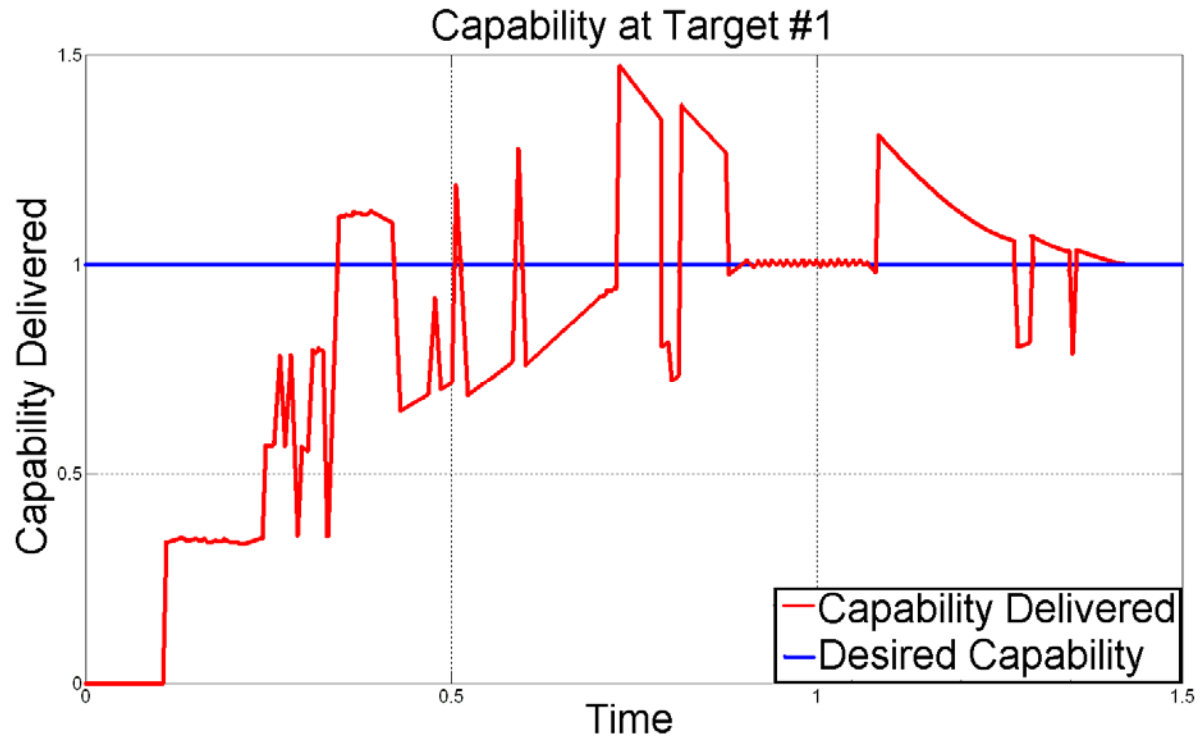

Figure 42: Test Bed #5 – Capability on Target #1 over time.

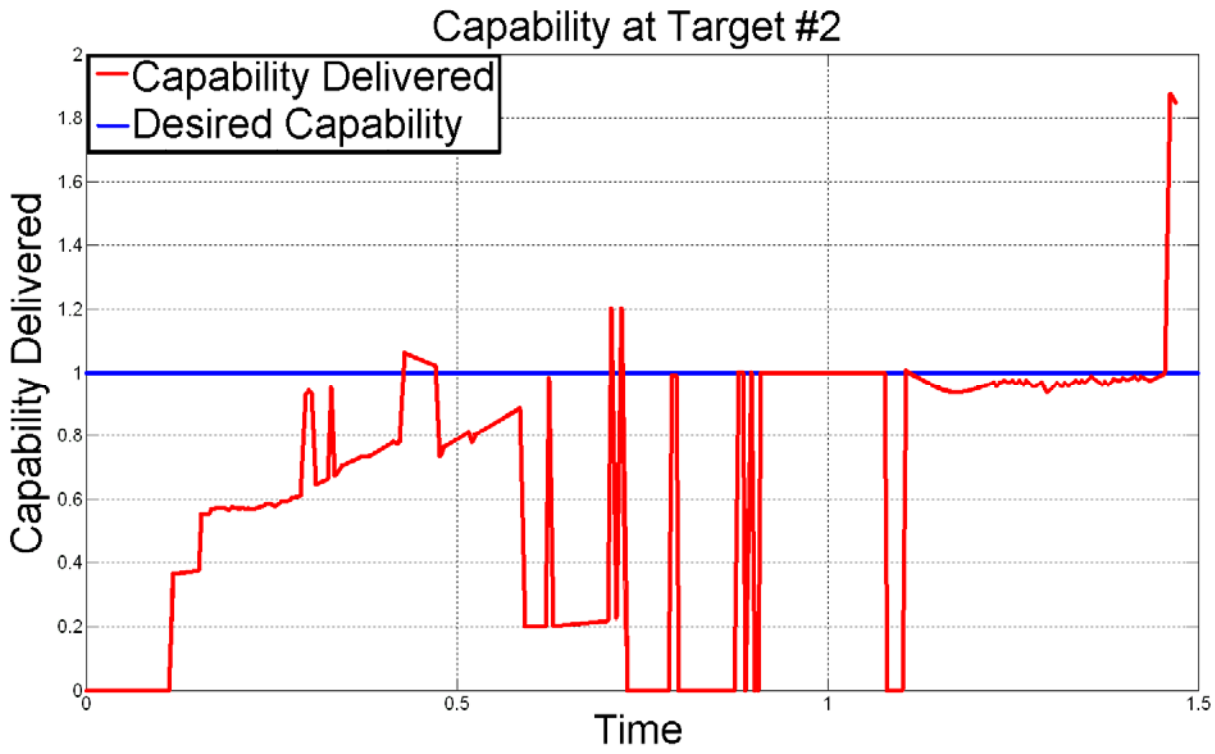

Figure 43: Test Bed #5 – Capability on Target #2 over time.

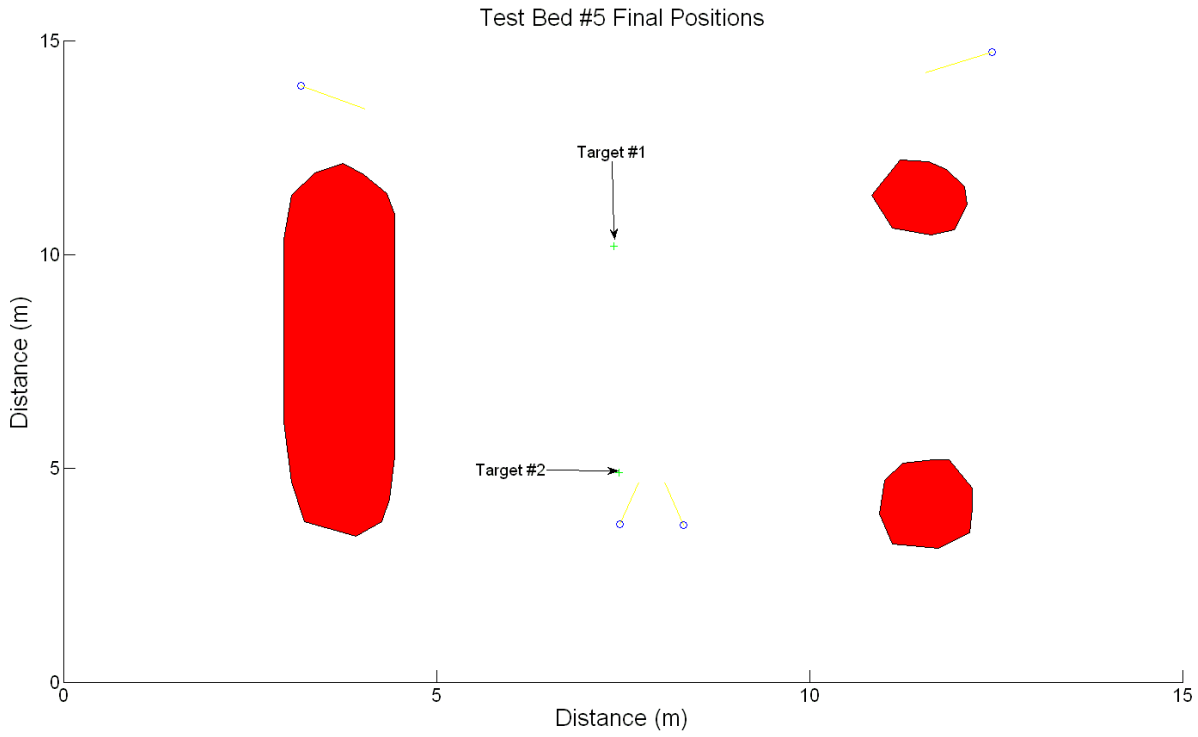

Figure 44: Test Bed #5 – Final position and orientation.

At first glance at Figure 44, it appeared that the robots failed to meet the assigned objectives. However, this was not the case. This was a perfect example of resource allocation by the swarm. Not in the traditional sense that it dedicated half its units to one target and half to the other, but that two of the robots were able to position themselves so that they were delivering capability on multiple targets. The desired capability on target #2 was achieved by robots #3 and #4 alone. Each of them delivered a capability level of 0.9, which was why the level of capability in Figure 43 was well over the desired level of 1. However, for target #1, all four robots were delivering capability. While each one was only able to deliver capability at a level of approximately 0.3 apiece, the four combined allow the desired capability to be reached. For this target there was a tradeoff between the pixel resolution and multiple perspectives. When dealing

with multiple targets, the mission often takes much longer to accomplish since all desired capabilities must be met simultaneously.

G. Conclusions

Overall the performance of the swarm in the five different scenarios was considered very successful. As one can see by the simulation and capability plots the robots were able to achieve the primary goal of delivering the desired level of capability to the target location(s). The swarm was also able to accomplish all of the secondary objectives. This was crucial to the overall performance of the swarm. While the primary objective had to be accomplished for the mission to be successful, it was the secondary objectives that allow the swarm to be considered for real-world use in the future.

There was one issue, however. Because of the nature of angular directions and bearing there was not a continuously smooth way to represent over 360 degrees of rotation. For our angular calculations it was necessary to determine which quadrant of the angular spectrum our robot was oriented towards. For example, if the robot was facing to the right (which was assumed to be 0°) it was necessary to change from a 0° to 360° frame of reference to a -180° to +180° so that there was not the discontinuity between 0° and 359°. The problem arose when a robot or target location was completely surrounded in all 4 quadrants. While the controller worked in test bed 3 for that particular case, times arise when the robots become confused in a sense. Because of the constant changing in degree reference frames, the robots can possibly become stuck trying to decide which direction to approach the target from. This was a simple mathematical problem that is currently being worked on.

Despite that small issue, the controller has been very successful over all. When running most of the simulations, the most time intensive thing was just finding the balance between the magnitudes of the different potential fields. Since some objectives obviously can have a higher priority, such as obstacle avoidance, it was important to calibrate the system so that the APF developed by the secondary controller for obstacle and robot avoidance was not overpowered by the other APFs. Once that balance was established though, the swarm was able to accomplish the scenarios with relative ease and fluidity.

8. Hardware Implementation

The second phase of the project involved hardware implementation of the algorithms. In order to implement the actual hardware, many real world technical issues that are not apparent in computer simulation had to be taken into account. These issues included vehicle kinematic constraints, vision system calibration, Bluetooth communications, and computer processing speed. For vehicular hardware, the Khepera II Mobile Robot, shown in Figure 45 was used.

Figure 45: Khepera II Mobile Robot.

The Khepera II is a miniature mobile robot designed to mimic the capabilities of larger robots used in various research and education venues. It measures only 70 mm in diameter and stands 30 mm high. Due to its small size, it is able to provide real world testing of various control algorithms without the necessity for an extremely large workspace. It features a Motorola 68331, 25 MHz processor and is propelled by 2 DC brushed servo motors, achieving a max speed of 1 m/s.[12]

In order to communicate wirelessly, the system incorporated a USB Long Range Bluetooth Dongle from Micro-Star International. This dongle was able to communicate with up to seven different devices at once and has a transmission range of over 100 m.[13] The Khepera II Mobile Robots were equipped with a Bluetooth Radio Turret developed by Road Narrows Robotics seen below.

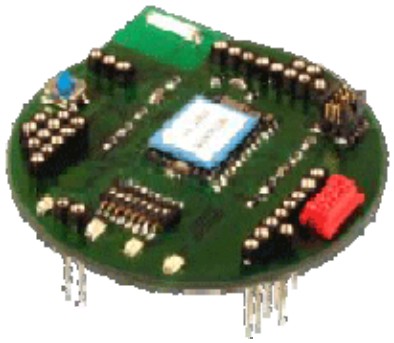

Figure 46: Khepera Bluetooth Radio Turret.

This was slightly different than the standard Khepera radio turret as it replaced the RS232 serial link between the robot and the PC and it did not use the KNet bus to communicate with the Khepera. In order to connect the turret to the PC, a baud rate of 115200 bits per sec was required. However, the connection between the turret and the Khepera operated at the baud rate 38400 bits per second. Therefore the Khepera was set to operating mode number 3 for serial communication at 38400 bps. Now that the controller could communicate with the individual

units of the swarm, the controller previously developed was adapted to the new real world environment. However, one aspect of simulation that still remained in this phase was the specific sensing capabilities. Due to the scale of the experiment, the Kheperas could not be mounted with an actual vision system. Therefore, the controller assumes that each unit has a virtual camera system as defined by the user and treats it accordingly. In the results section, one will be able to see how the individual virtual camera systems were simulated onto a real world robotic platform.

A. Computer Vision Localization

The first major step in the transition from MATLAB Simulation to a real-world implementation was to develop a way to enable the robotic units to localize themselves in the provided environment. Ideally, in the operational environment, GPS would be used to position and track the swarm units in a much larger environment. However, due to the scale of the Khepera robots and the environment, the controller was unable to use GPS positioning as the resolution would not be sufficient. This investigation, therefore, used a standard USB Webcam that was positioned above the created test bed to locate each robot, as seen in Figure 47. A simple webcam was selected because of its ease of integration into MATLAB and its low requirement for PC processing resources.

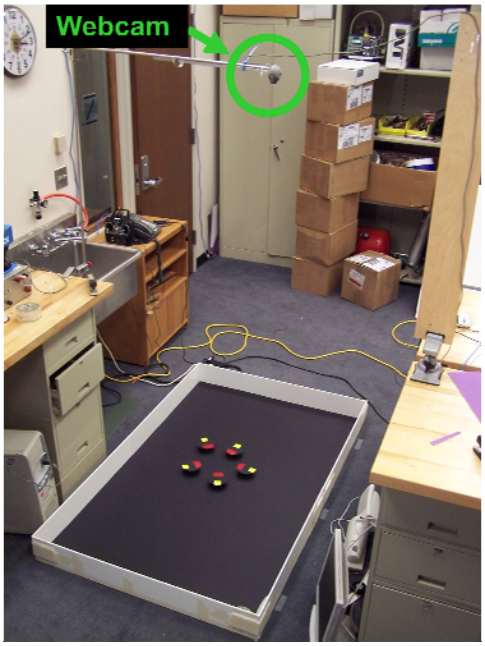

Figure 47: Test Bed Environment with Webcam installed.

In order for the computer to recognize the robots, each was equipped with a color coded platform that rests on top of the unit. This platform was black, which matches the surface of the environment, and had a red colored area on the front of the robot and a yellow area towards the rear of the robot. Every time the vision system took a picture of the environment, code was developed to locate each color "blob" in the image. Two matrices were created containing the location for all of the red and yellow pixels. From this data, the centroid locations for the red and yellow values were calculated and can be used to determine the robot's position and orientation. The centroid of the red area was used as the actual location of the camera on each robot. This was important because it was not located on the single wheel axis of the robot, and it allows compensation for the non-holonomic nature of the Khepera robots. This will be discussed in detail in the following "Kinematic Constraints" section. The centroid of the yellow area was used to determine the bearing of the robotic unit. Ideally, in the real world, each unit could be equipped with some sort of electronic compass to determine robot heading.

Now that the location could be calculated for a single unit, the controller had to be able to calculate the location and orientation for multiple units and be able to track and discern them from one another. This created inherent problems, as the search program had to be adjusted to find groups of red areas instead of arbitrarily finding all the pixels. MATLAB's "blob analysis" source files were incorporated into the search so that groups of similar pixels could be located and set a certain threshold value to ensure they were of sufficient size and not simple outlier data. Each of these "blobs" could then be assigned a designator and tracked independently of one another. Then the controller could successfully locate and track a swarm of any size with units equipped with the created color coded platform. These locations could then be inserted into the already developed controller and used to calculate the units' X and Y positions and θ headings.

One of the problems encountered, was the change in axis from the standard coordinates to the pixel reference frame. The standard coordinate frame has the origin in the lower left corner with the positive y-axis in the vertical direction and the positive x-axis horizontally to the right. The pixel reference frame is slightly different. In this reference frame the origin is actually in the upper left corner with the positive y-axis in the downward vertical direction, opposite of the standard coordinate frame. This was because each pixel location was read as the row and column with row increasing as one moved down the plot and column numerically increases as one moved to the right. A visual representation can be seen in Figure 48.

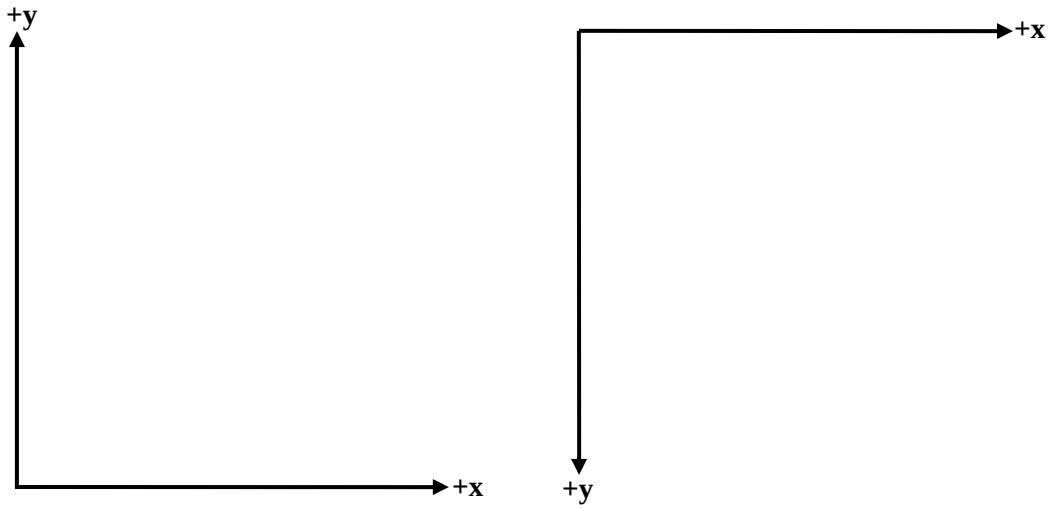

Figure 48: Comparison of Standard Coordinate Frame (Left) and Pixel Reference Frame (Right)

The solution to this problem did not involve the changing of any of the capability functions. It was solved by manipulating the interface between the robots and their environment. This change had to be taken into account whenever calculating bearing and desired directional changes. As for movement in the physical environment, the only real obstacles to overcome were generated by the kinematic constraints of the vehicle.

B. Kinematic Constraints

With the ability to locate every unit and calculate its current heading from the vision system, this information was plugged into the previously developed controller from section 6 with a few minor changes. The controller then provided each unit with the desired direction of motion as well as desired velocity. The challenge, however, arose in having the units move in the desired direction. Due to the non-holonomic nature of the Khepera robots, they cannot be simply given a direction of motion and be expected to move in that direction. There were ways to work around this, however.

For each of the units it was assumed that the virtual camera system was located on a control point on the front of the robot. By placing this point at a location off of the common wheel axis it was possible to provide wheel velocity commands to the wheels of the robot that will allow the control point to move in any direction at any given time. Therefore, a non-holonomic vehicle can be turned into a platform that, when provided the right commands, can allow a control point on the front of the robot to behave holonomically. This was the reason the centroid of the red colored area was used as the robot position despite the fact it was not in the middle of the robot. This proved crucial to the success of the real world implementation.

In order to move the control point in a holonomic manner, a Jacobian matrix could be developed to control the differentially driven mobile robot. By having the control point define the output state for position feedback and artificial potential fields, the platform could be supplied with commands to carry out the desired motion. A schematic of the robot can be seen in Figure 49.

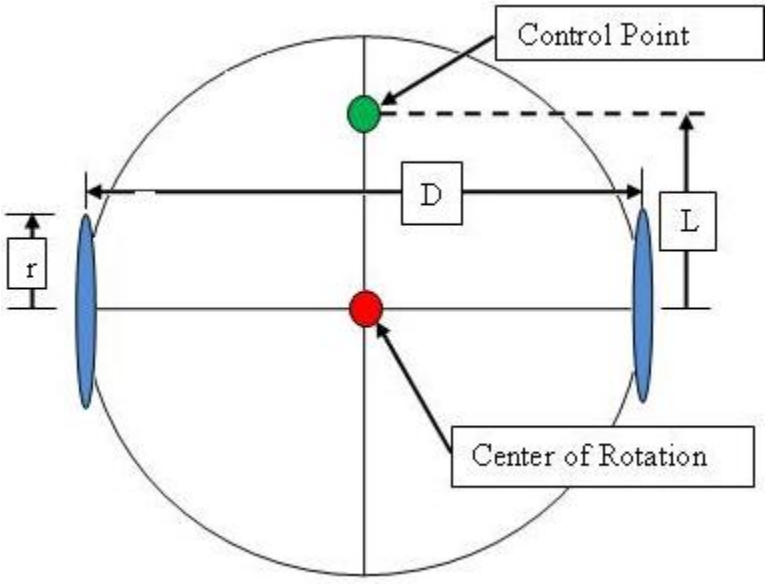

Figure 49: Khepera Schematic with Measurements and Control Point

Given this configuration, a Jacobian was developed relating wheel velocities to control point motion. If ω_1 and ω_2 are the wheel angular velocities then we can establish the following equations.

$$\vec{v}_c = \dot{\vec{p}}_c = \begin{bmatrix}\cos\theta \\ \sin\theta\end{bmatrix}\frac{r(\omega_1 + \omega_2)}{2}$$

Equation 8.1

$$\dot{\theta} = \frac{r(\omega_1 - \omega_2)}{d}$$

Equation 8.2

In Equation 8.1, $\vec{v}_c$ was the velocity at the center of rotation for the unit and $\dot{\theta}$ was the angular velocity. Using these two equations we were able to develop an equation for the velocity at the control point, $\vec{v}_{cp}$.

$$\vec{v}_{cp} = \vec{v}_c + \dot{\theta} * L\begin{bmatrix}\cos(\theta + 90) \\ \sin(\theta + 90)\end{bmatrix}$$

Equation 8.3

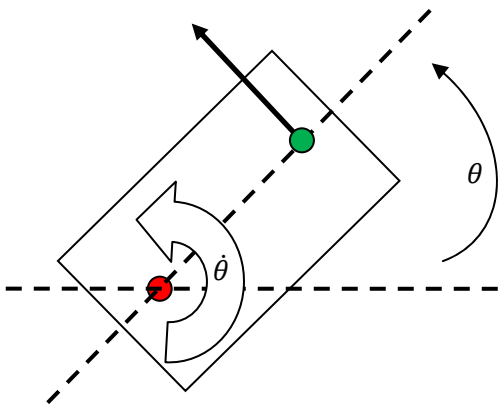

Figure 50: How the Angular Velocity Affects Control Point Motion

Figure 50 shows how the angular velocity of the robot about its center of rotation has a tremendous effect on the velocity of the control point. Although this complicates the calculations, this was the very principle that allows the control point to act holonomically. Since the robot can translate and rotate at the same time, the control point can move linearly in any direction based solely on the commands provided to the robots.

Based upon Equation 8.2 and Equation 8.3, the velocity command were separated into an X, Y and θ component. This can be seen in Equation 8.4.

$$\begin{bmatrix} \dot{x}_{cp} \\ \dot{y}_{cp} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \dot{x}_c + L\dot{\theta}\cos(\theta + 90) \\ \dot{y}_c + L\dot{\theta}\sin(\theta + 90) \\ \frac{r}{d}(\omega_1 - \omega_2) \end{bmatrix}$$

Equation 8.4

In this equation, one can see how the specific velocity components are affected not only by the dimensions of the platform but also the angular and linear velocity of the center of rotation. Substituting portions of Equation 8.1 and Equation 8.2 into this equation a matrix was left that was based only upon the wheel angular velocities and the current heading, as seen in Equation 8.5.

$$\begin{bmatrix} \dot{x}_{cp} \\ \dot{y}_{cp} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \frac{r}{2}(\omega_1 + \omega_2)\cos\theta - L\frac{r}{d}(\omega_1 - \omega_2)\sin(\theta) \\ \frac{r}{2}(\omega_1 + \omega_2)\sin\theta + L\frac{r}{d}(\omega_1 - \omega_2)\cos(\theta) \\ \frac{r}{d}(\omega_1 - \omega_2) \end{bmatrix}$$

Equation 8.5

This equation has provided the desired linear and angular velocities in terms of the heading, θ, and the wheel angular velocities. The next step was to pull the wheel angular velocities out of the matrix so that they may later be solved for. This is shown in Equation 8.6.

$$\begin{bmatrix} \dot{x}_{cp} \\ \dot{y}_{cp} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \frac{r}{2}\cos\theta - L\frac{r}{d}\sin(\theta) & \frac{r}{2}\cos\theta + L\frac{r}{d}\sin(\theta) \\ \frac{r}{2}\sin\theta + L\frac{r}{d}\cos(\theta) & \frac{r}{2}\sin\theta - L\frac{r}{d}\cos(\theta) \\ \frac{r}{d} & \frac{r}{d} \end{bmatrix} \begin{bmatrix} \omega_1 \\ \omega_2 \end{bmatrix}$$

Equation 8.6

Since the angular velocity at the control point is not a concern, it can be eliminated from the bottom portion of the matrices. Now a 2x2 matrix is left that was actually the Jacobian for the system and the current equation can be seen in Equation 8.7.

$$\begin{bmatrix} \dot{x}_{cp} \\ \dot{y}_{cp} \end{bmatrix} = \begin{bmatrix} \frac{r}{2}\cos\theta - L\frac{r}{d}\sin(\theta) & \frac{r}{2}\cos\theta + L\frac{r}{d}\sin(\theta) \\ \frac{r}{2}\sin\theta + L\frac{r}{d}\cos(\theta) & \frac{r}{2}\sin\theta - L\frac{r}{d}\cos(\theta) \end{bmatrix} \begin{bmatrix} \omega_1 \\ \omega_2 \end{bmatrix}$$

Equation 8.7

Now, the resulting velocity at the control point can be determined in terms of the X and Y components based upon what wheel velocities provided. This was, however, the exact opposite of what was desired. Because the controller was able to deliver desired velocity in X and Y components, the desired wheel velocities had to be calculated to provide to the robots so that they may move in the desired direction. Therefore the inverse of the Jacobian matrix was taken and manipulated so that when provided a desired velocity, the necessary wheel commands to move in such direction could be calculated. This is shown in Equation 8.8.

$$\begin{bmatrix} \omega_1 \\ \omega_2 \end{bmatrix} = \begin{bmatrix} \frac{r}{2}\cos\theta - L\frac{r}{d}\sin(\theta) & \frac{r}{2}\cos\theta + L\frac{r}{d}\sin(\theta) \\ \frac{r}{2}\sin\theta + L\frac{r}{d}\cos(\theta) & \frac{r}{2}\sin\theta - L\frac{r}{d}\cos(\theta) \end{bmatrix}^{-1} \begin{bmatrix} \dot{x}_{cp} \\ \dot{y}_{cp} \end{bmatrix}$$

Equation 8.8

It is this final equation that is the backbone of the solution to the kinematic constraints. The desired unit velocities could now be supplied, calculated by the controller, to a program that was able to convert them into angular wheel velocities. These velocities were then sent to the robot units using serial communication, based upon such commands, the control point for each of the units moved in the desired direction.

C. *Avoiding Obstacles*

The obstacle avoidance portion of the controller proved to be drastically different in the hardware configuration than that of the previous MATLAB simulation. In the first semester's work, the user created a matrix of obstacles for the environment and the controller was aware of not only the shape and size of the obstacle, but also the location for every single vertex. Since the swarm was being designed to operate in an unknown environment, it was important to transition from a reliance on the controller to direct the units around the obstacles towards a solution that allowed the robots to react independently when encountering an obstacle. In all of the real-world test solutions, the overhead computer vision system had no knowledge as to the location of any of the obstacles. The only information the controller received throughout the entire real world testing process was the X and Y location and heading θ of the individual units as well as any information that was returned by the actual sensors on the robot units themselves.

Since the controller was relying on the robot to react independently with the environment, the use of the eight Infra-red proximity and ambient light sensors on each of the units were incorporated. The configuration of the sensors can be seen in Figure 51.

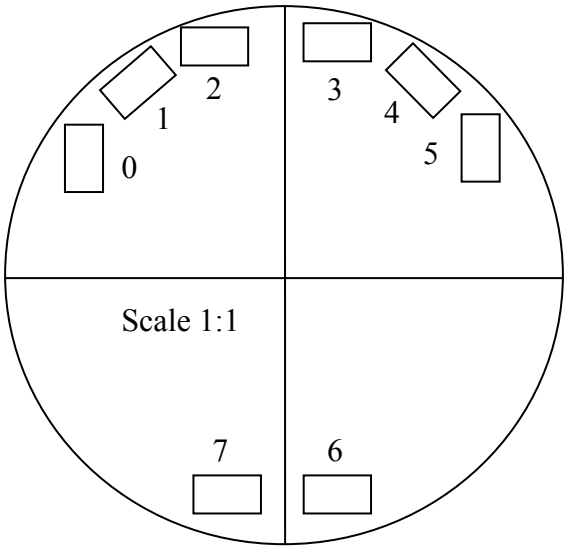

Figure 51 : Schematic of Khepera II Mobile Robot with Sensor Orientations

Each of the eight sensors is a TCRT1000 reflective optical sensor from *Vishjay Telefunken* permitting two measurements. The first is a measurement of the normal ambient light. This measure is made using only the receiver part of the device; hence, it does not emit any light using the emitter. A new measurement is made every 20 ms in sequential order with a 2.5 ms separation between each sensor. The second measurement taken is the light reflected by obstacles. In this measurement, light is emitted using the emitter and the returned value is the difference between the ambient light reading and the reflected emitted light reading. These outputs are then converted into a 10 bit digital value, which can be read through the serial communications with the robot. The rated performance of the sensors can be seen in Figure 52-Figure 53.[14]

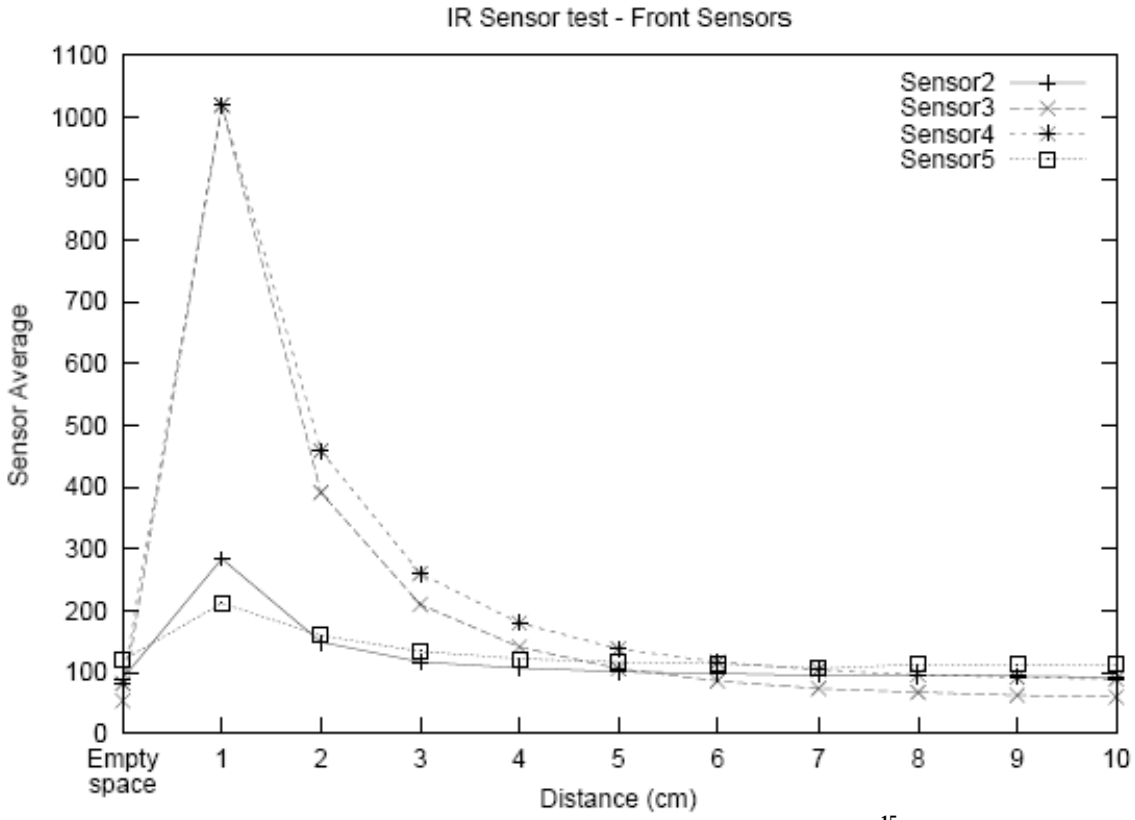

Figure 52: White Paper Detection Test (Front Sensors)[15]

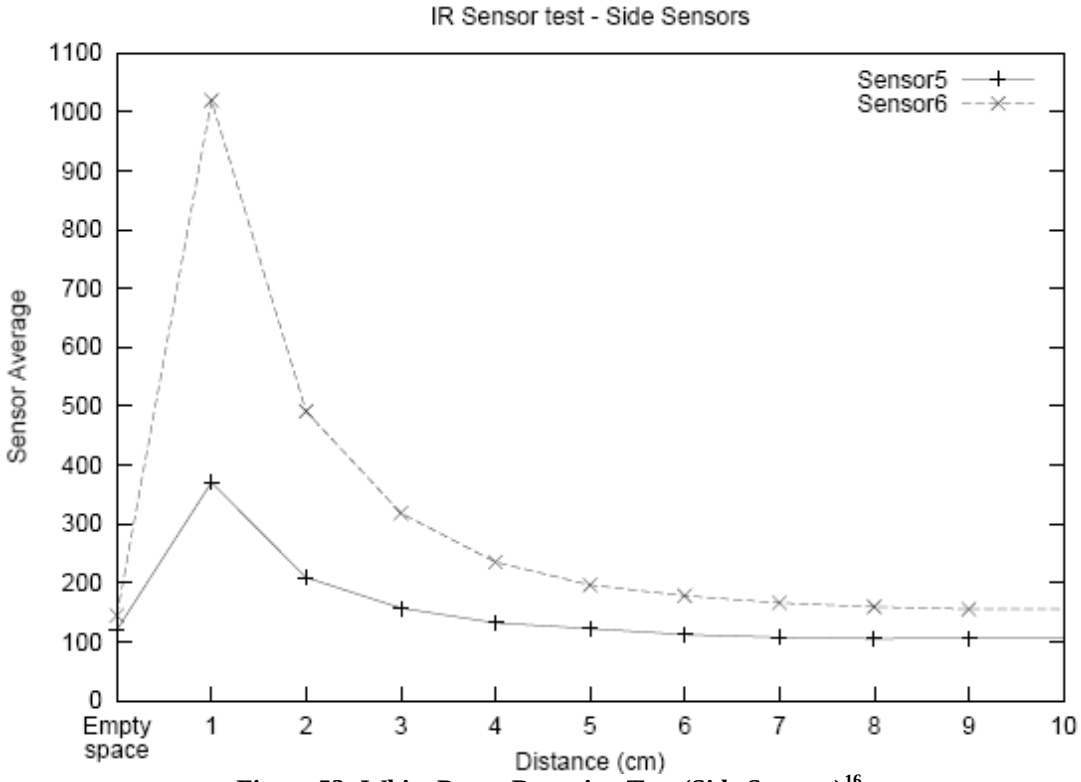

Figure 53: White Paper Detection Test (Side Sensors)[16]

It is important to notice the sharp drop off in the sensor readings after distance increases over three centimeters. This proved to be a crucial limiting factor in how the robot reacted to its environment as the robot did not often sense the obstacle until within centimeters. This delay forced an adjustment in the repulsive force component of Equation 6.1 as a more immediate reaction was required in order to avoid the obstacles. Therefore, when a secondary task such as obstacle avoidance was encountered, that had a greater impact on the desired motion than before.

Whenever one of the eight sensor returned a reading greater than 150, a repulsive vector was sent to the task space controller in a direction dependent on which sensor was tripped. This vector was added to any others determined by the task space controller. Since this vector was in the unit reference frame, it must then be compared to the unit heading so that it may be converted to the environment's reference frame for the final controller to utilize. Figure 54 displays the repulsive vectors in the robot unit reference frame.

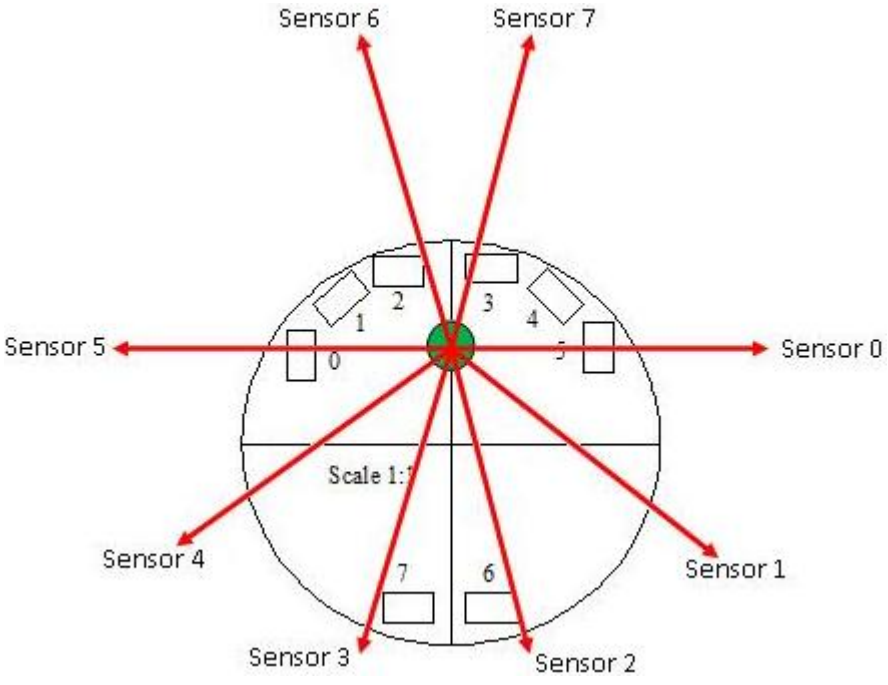

Figure 54: Repulsive Vectors for Each Sensor

It was important to note that if multiple sensors were triggered, the repulsive vector from each triggered sensor would be combined into one overall vector for obstacle avoidance. It was this vector alone that would be sent to the task space controller. This would then be combined with the desired heading and velocities as determined by the primary controller and would result in a desired motion for the unit to simultaneously accomplish all tasks.

D. Avoiding Other Robots and Objective Location

Theoretically, in order to avoid other robots one could have been able to use the same range sensors that were used for obstacle avoidance. However, a major problem was encountered when attempting this. It turns out that the range sensors were very dependent on the reflective surface being detected. Figure 52 and Figure 53 were conducted on a white paper test which had significantly better results than other surfaces. When detecting other robots, due to the chassis surface, very poor returns in the range sensor data were experienced.

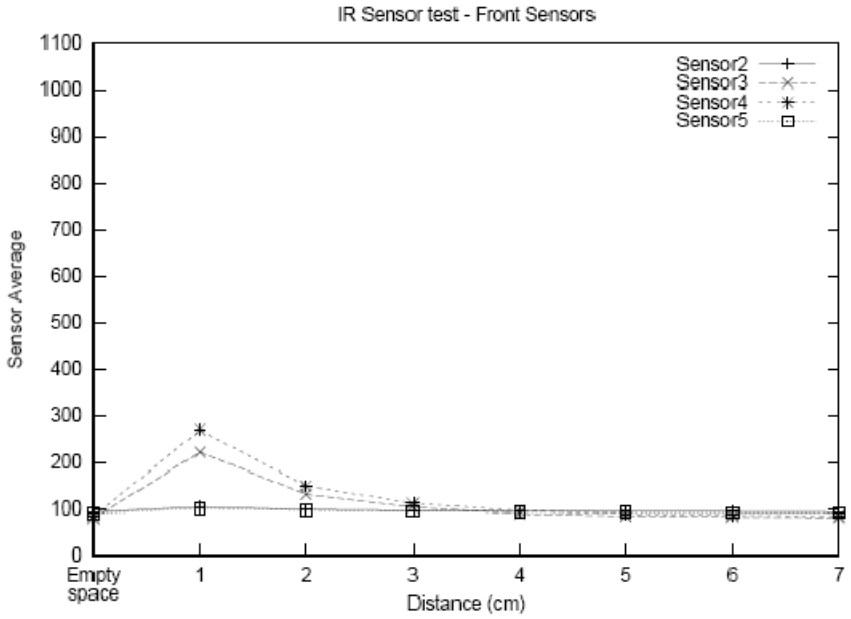

Figure 55: Other Robot Detection Test (Front Sensors)[17]

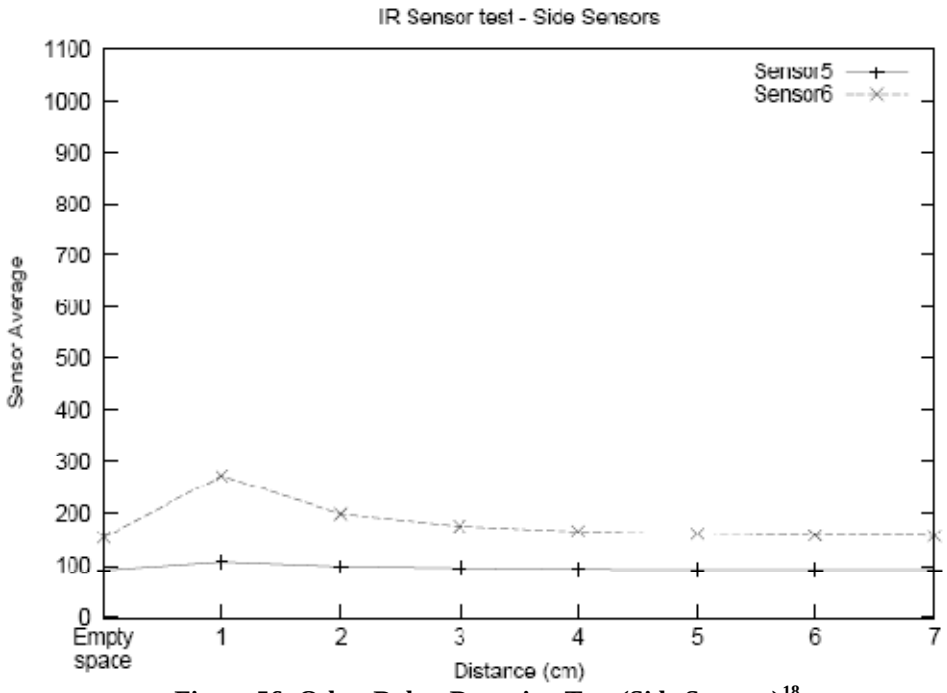

Figure 56: Other Robot Detection Test (Side Sensors)[18]

As seen in Figure 55 and Figure 56, the ability for the sensors to detect other robots was very limited. Because of this, the concept used for obstacle avoidance was very similar to that used in the original computer simulation. Based upon each unit's physical location, the controller runs a search program that calculates the distance between every robot. If any were within the designated "Safe Zone", the computer would automatically generate a repulsive vector in the opposite direction. The sensors were still active, and should something go wrong with the controller such that it does not detect other robots, the sensors would still treat other robots as obstacles should they come close enough for detection. This was somewhat of a back up, but never happened due to the robustness of the controller.

This technique was actually ideal since the proposed missions for this controller are focused on reconnaissance, and inherently incorporate stealth into the platforms. Depending on the units being used for the mission, it was possible that the units' obstacle avoidance capabilities may not even be able to detect the other robots. This could be due to their shape, size, color, or

even their profile on the ground. Because the only thing the controller did know was their specific location, the controller can then use this information to avoid collisions even though each robot could never actually see another.

Target location avoidance was handled in the same way. Since the target could be just an empty space that needed to be investigated, the robot cannot rely on its sensors to repel from the target. This becomes even more evident for some of the potential operations of the swarm in the real world such as landmine detection using a magnetic sensor. If a swarm was sent to investigate an area for potential landmines, the robotic platform's obstacle avoidance sensors cannot be used to avoid the target location since the potential landmine is buried underground. Therefore, the same search program used in robot avoidance was utilized. Instead of comparing distances to other robots, the program compared the distance of every robot to the target location and could create a repulsive vector should the "Safe Zone" be violated.

9. Real World Test Results

After adapting many of the simulation programs to account for various attributes of the hardware, testing in a real environment began. A series of test beds were set up that would test four fundamental control concepts:

1. Integrated obstacle avoidance.
2. Convergence on target location based on capability alone.
3. Approach to the target location from different directions.
4. Ability to adapt to a heterogeneous swarm without any change to basic controller.

In order to effectively test these abilities independently and in combination, this research started with a basic test bed that would ensure that each individual robot would operate correctly and then gradually progressed through the test beds, with each becoming more complex.

A. Test Bed #1

In this test bed, a single robot moved through the environment to provide specified capability at a single target point, in the presence of a simple obstacle. The delivered capability was computed as if the robot were equipped with a 35 mm camera lens system as seen in Figure 9: This is a 3D representation of the capability delivered by a 35 mm Lens . The purpose of this simple test bed was to ensure that each individual unit would be able to act appropriately before being implemented into a larger swarm.

As the robot moved through the environment, it approached an obstacle that was undetectable by the camera vision system (as discussed in section 8.C). The robot successfully used its eight IR range sensors to detect the obstacle, and a repulsive force was generated accordingly. This force, combined with the desired direction of motion from the primary controller, created a resultant vector and allowed the robot to drive around the obstacle without striking it. Once around the obstacle, the robot took the shortest path to one of the locus of points at which it could deliver appropriate capability to the target location. For this trial, the desired capability was 0.9, indicating that the unit had to be able to deliver 90% of its maximum sensing capability to the target location. The resulting capability plot over the course of the test run can be seen in Figure 57.

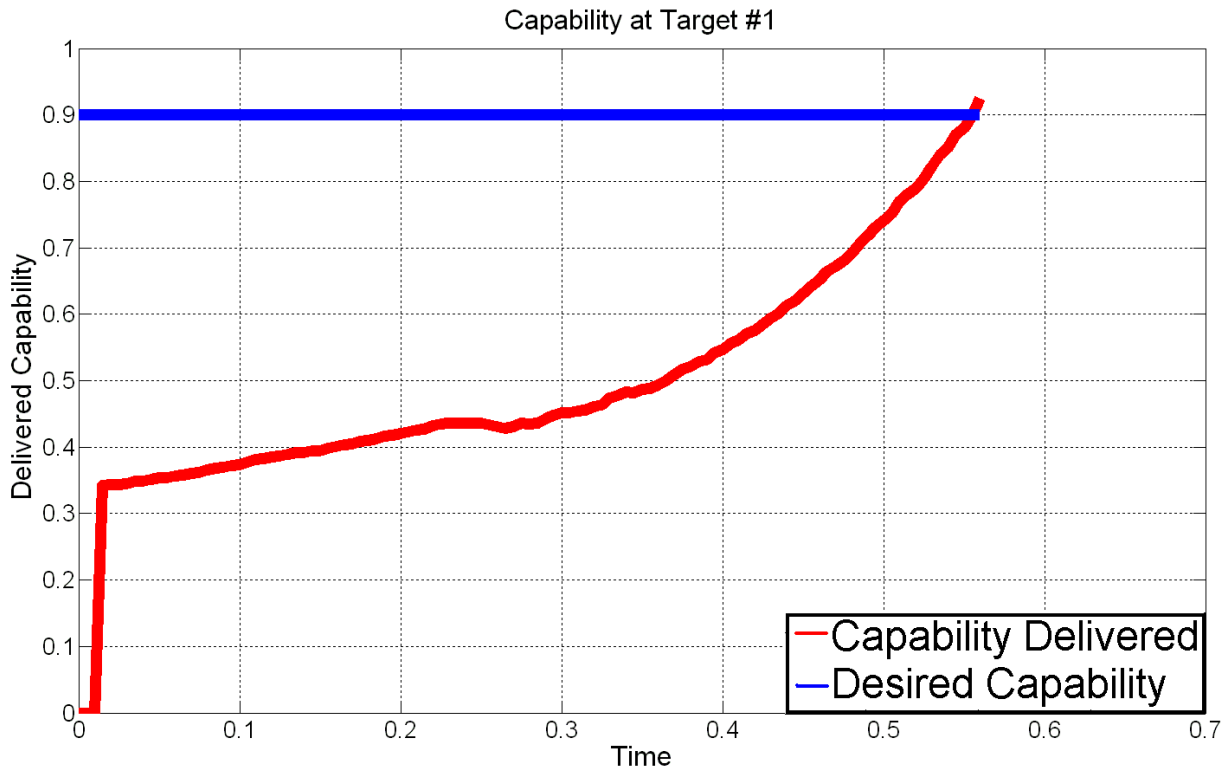

Figure 57: Test Bed #1 Delivered Capability

Looking closely at the shape of this graph, one can see the sharp spike in capability as the virtual, on-unit camera acquired line of sight on the target location. Again, in all of these experiments, the virtual camera was assumed to be able to rotate freely on the robotic platform. This allowed the robot to turn and move without the camera losing sight of the location.

Once the virtual on-unit camera acquired the target, the robot moved towards the target location, gradually increasing the delivered capability along the way until the desired 0.9 capability was delivered. This trial was very successful, as not only did the robot deliver the needed capability, it also avoided all obstacles in its path along the way.

B. Test Bed #2

This test was designed to apply the basic primary controller functions to a swarm of five robots in a simple, obstacle free environment. The purpose of this test was to ensure that the controller could handle localizing and giving commands to multiple robots, and that the units would be able to act accordingly. A program was written that would search each frame of video captured by the overhead vision system for each of the robot positions. Because the robots could not be uniquely identified by visual information alone, the resulting localizations had to be correlated to past locations in order to determine the robot to which each set of color blobs corresponded. Then, the system would reassign the specific serial communication port to the appropriate robot. If serial communication was not an issue, the controller could just arbitrarily assign each robot an identifier and calculate the desired motion for it. However, because each robot has a specific serial port it must use, the controller has to be aware at all times which robot is which. In this experiment, the five robots began dispersed radially from the target location, and they converged appropriately without running into one another. The initial set up can be seen in Figure 58 and Figure 59. Each unit was "equipped" with an identical virtual camera with a 35 mm lens, as in Test Bed #1.

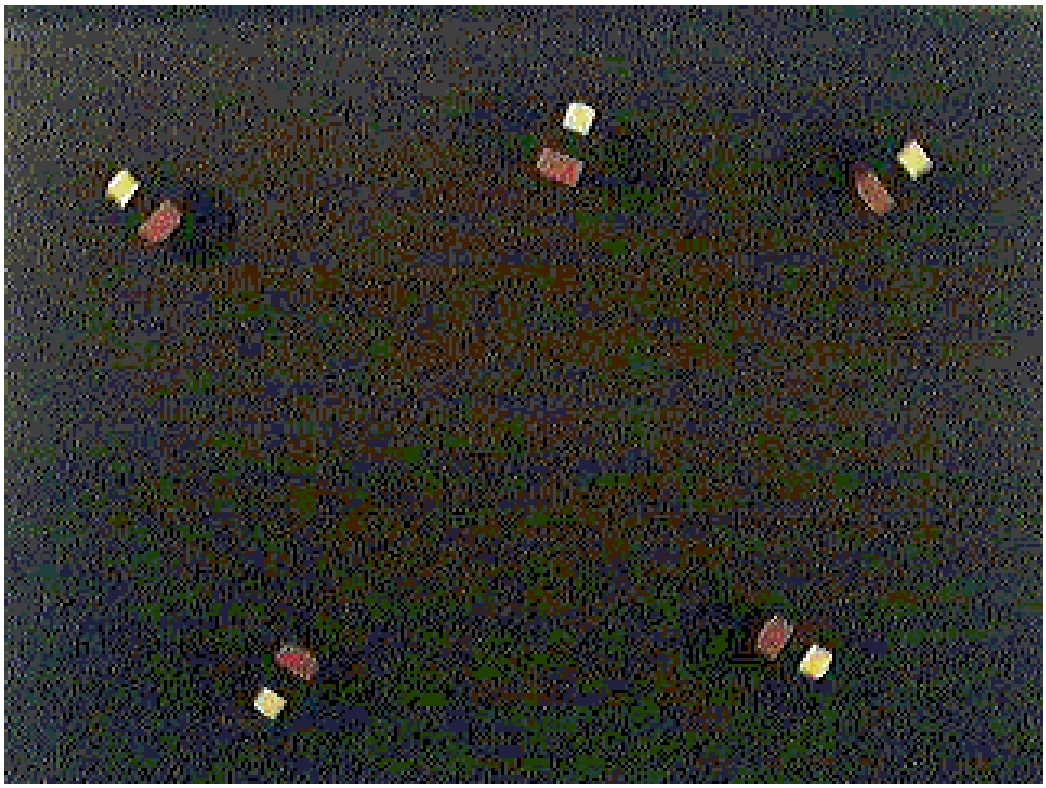

Figure 58: Camera View of Initial Set-up

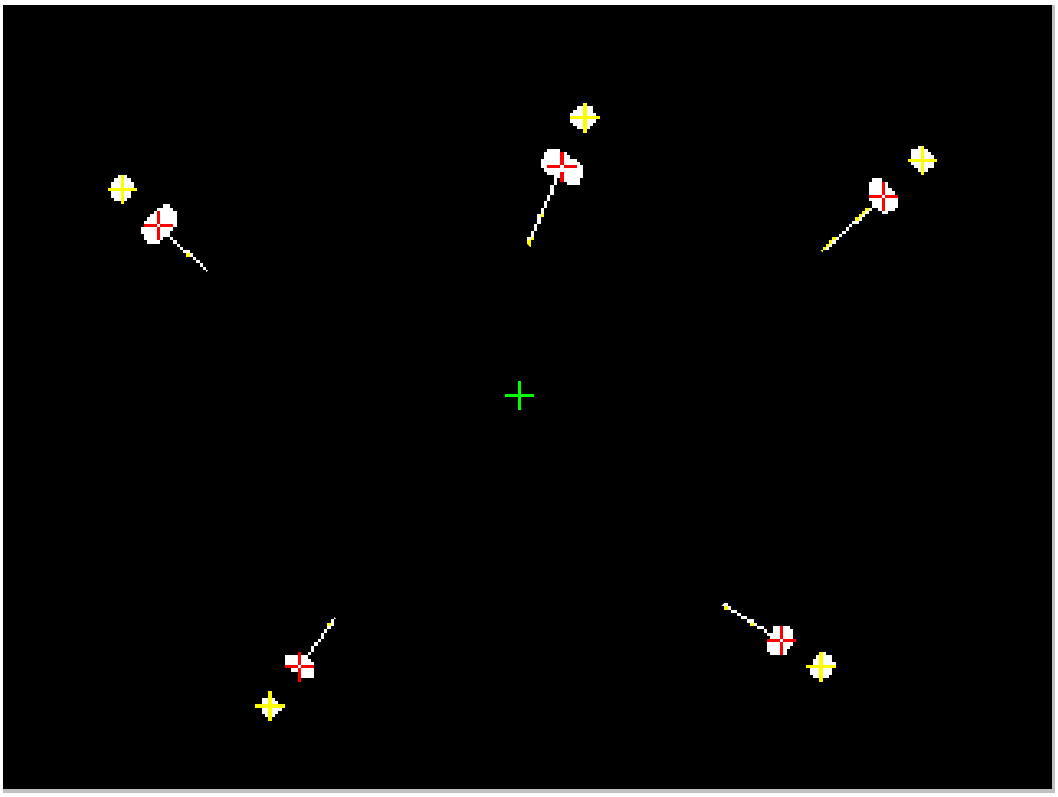

Figure 59: Binarized Computer Image of Initial Set-up

In the initial camera view (Figure 58), one can see the color coding on each of the robots. This correlates to the data visible in Figure 59, which indicates the location of the red and yellow areas. The line protruding from each robot's color blob set shows the heading of the robot, which was calculated from the blob locations for each robot. The base image in Figure 59 is binarized in the sense that it was purely black and white when originally computed. All of the other colored marks and lines were added post swarm identification. Each red and yellow cross represents the respective centroid for that colored area. The green cross in the center is the target location for this test bed.

Since this test bed had no obstacles, the swarm immediately converged directly toward the target location (as limited by unit kinematics). Due to their initial spread, it wasn't until they approached the target that they had to worry about colliding with one another, and they were successful at avoiding all collisions.

The desired level of capability for this test bed was 4.75 at the target location. As there are five units, the maximum deliverable capability to one target is 5.00, indicating that this test required all robots to cooperate in order to deliver the desired capability. As they converged on the target location, the robots successfully oriented themselves and their virtual cameras and carried out their assignment. The final poses can be seen in Figure 60 and Figure 61

Figure 60: Camera View of Final Position

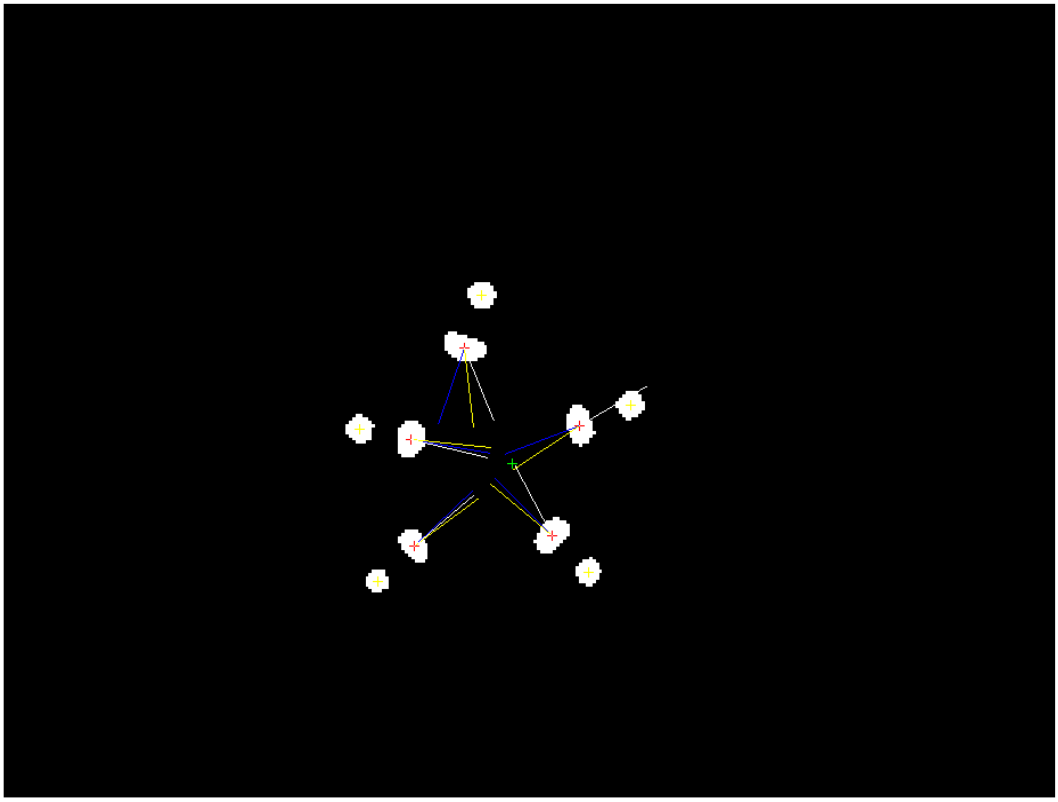

Figure 61: Binarized Computer View of Final Position

A legend for all of the color schemes in the final position can be seen in Figure 62.

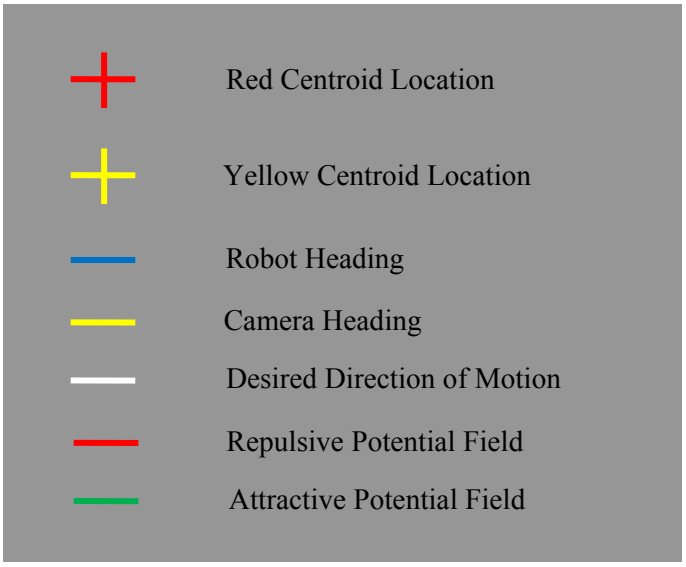

Figure 62: Controller Image Legend

As one can see in the previous images, the swarm was very successful at closing in on the target location and delivering the appropriate capability. The delivered capability plot for this test bed can be seen in Figure 63.

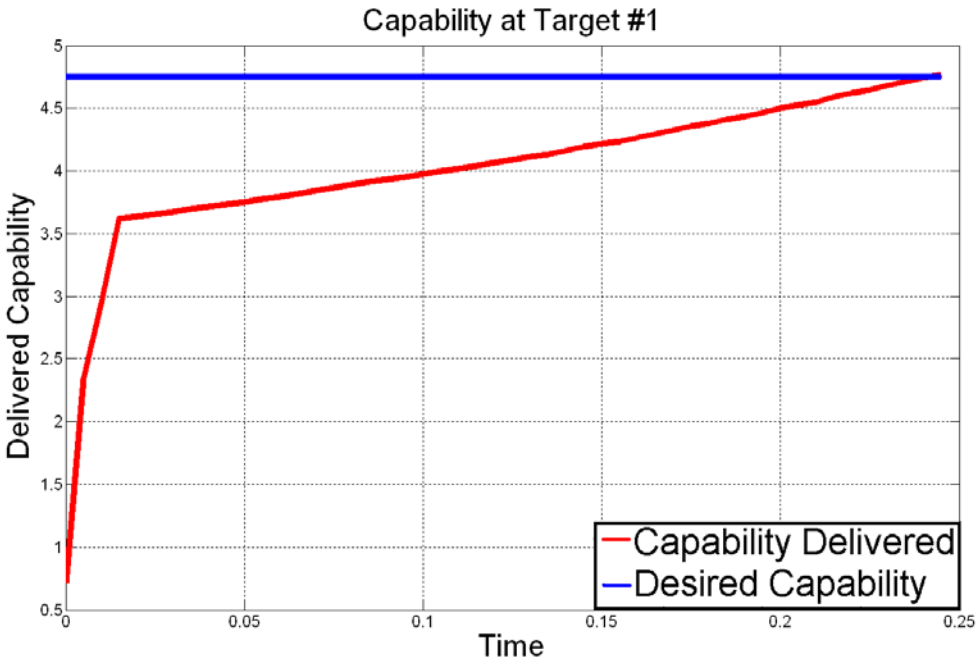

Figure 63: Test Bed #2 Delivered Capability

Overall this test was very successful. Once again, in the capability plot one can see the initial surge in capability as the virtual, on-unit camera systems achieved line of sight and continued to track the location of the target position. With the virtual cameras on target, the robots slowly crept forward until they were at a distance which allowed them to deliver the desired capability.

C. Test Bed #3

This test bed is similar to Test Bed #2 except that there are now obstacles placed throughout the test environment. The units are still homogeneous with a virtual 35 mm lens system. The swarm was still required to converge on the target location (based on desired capability), but the units also had to achieve obstacle avoidance. Any collision with an obstacle was taken as a failure of the test, even though such collisions were not necessarily catastrophic. The initial set up can be seen below in Figure 64.

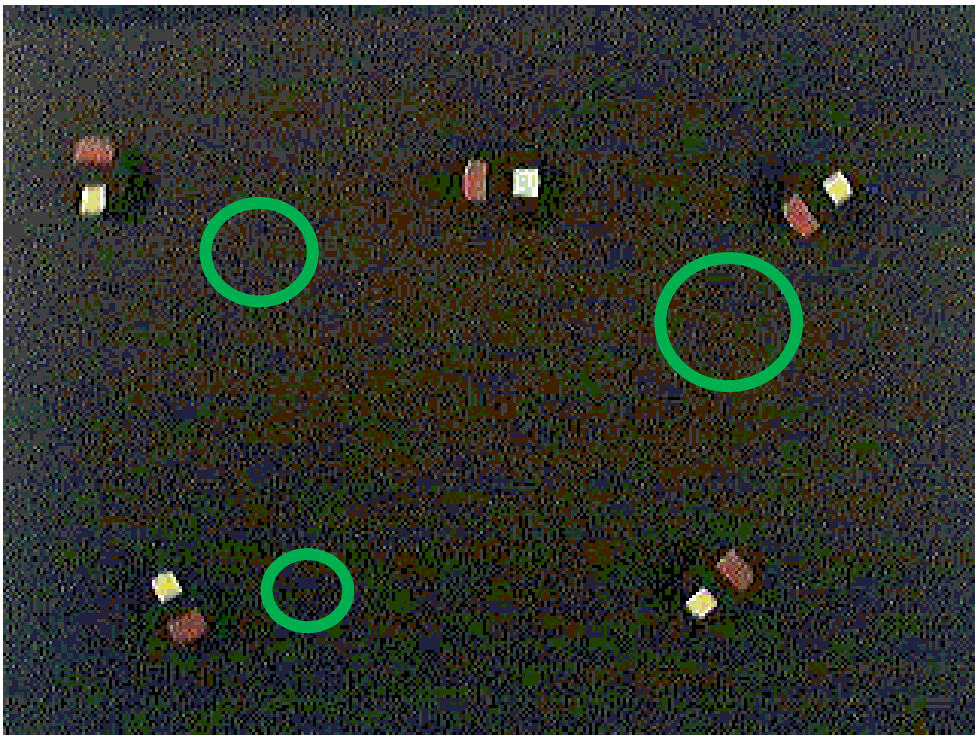

Figure 64: Camera View of Initial Set-up

Although it appeared that there were no obstacles in this environment, the tops to all of the obstacles were black; the exact same color as the environment floor. This was done for two reasons. First, it simplified the color thresholding for the overhead computer vision system. By only having two colors to track, one could expand the threshold levels for the system and generate a much more consistent and robust robot tracking scheme. Second, there was no need for the overhead camera to see the obstacles, as the robots were to use their IR sensors to provide environmental feedback data. Even though the top of the obstacles were black, the sides were white, as this is the best color for the IR sensors to detect. Green circles have been drawn on both maps which represent the approximate locations of all the obstacles.

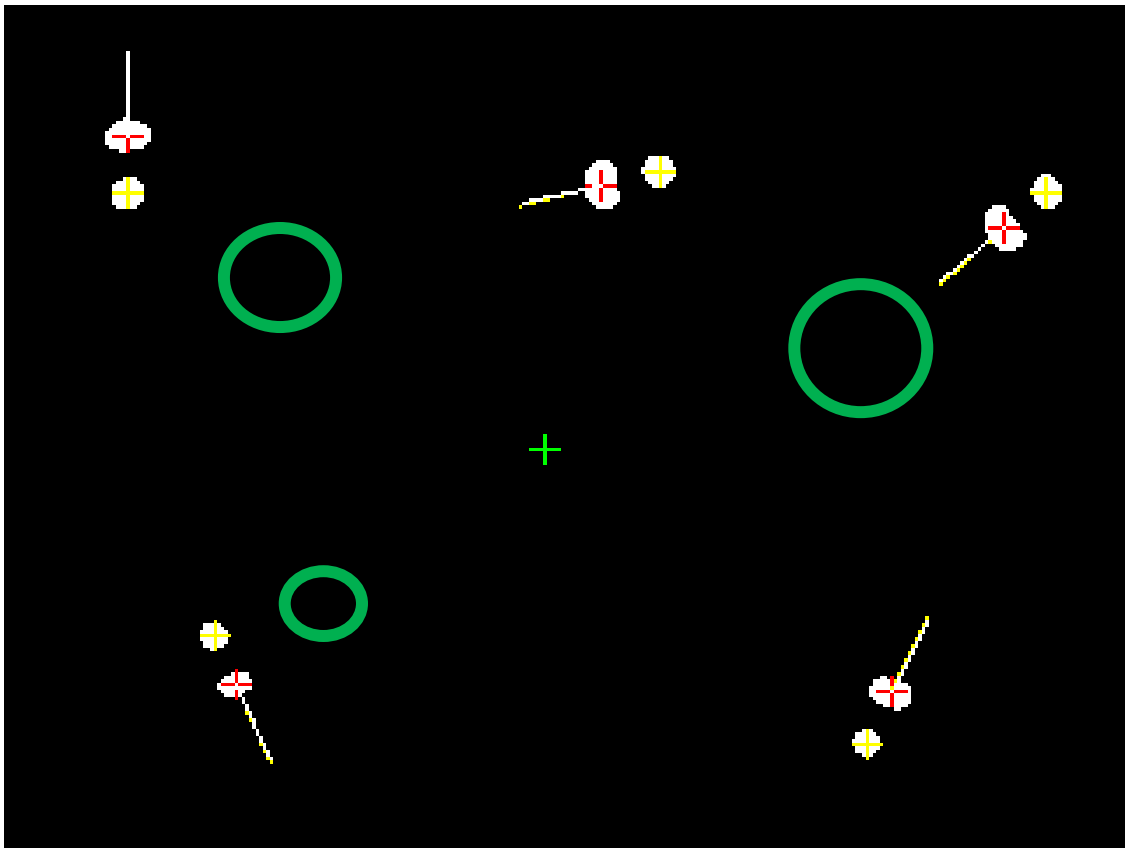

Figure 65: Binarized Computer Image of Initial Set-up

Again, this is a very similar set-up to test bed #2; however, since three of the five robots now obstacles in their way heading towards the target, they must be able to carry out avoidance maneuvers to reach the end goal. This trial was also very successful. The final positions can be seen in Figure 66 and Figure 67.

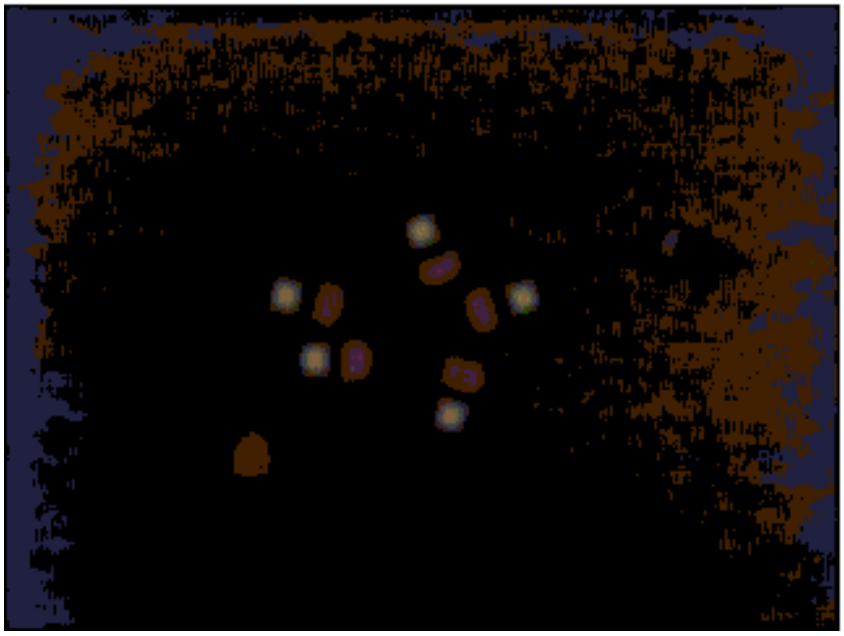

Figure 66: Camera View of Final Position

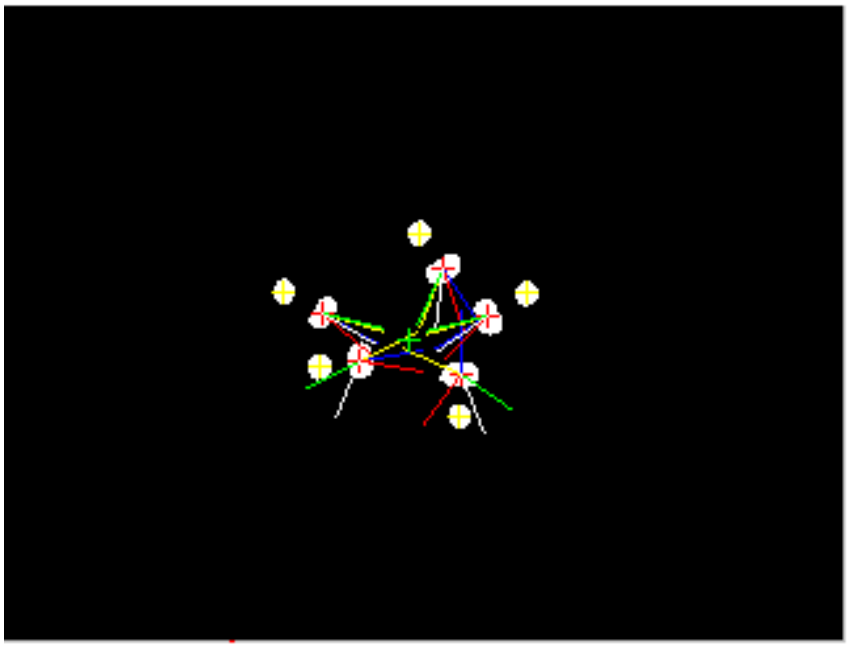

Figure 67: Binarized Computer Image of Final Position.

In both of the previous images, one can see how the swarm units have once again surrounded the target area and have delivered the appropriate capability. Refer again to Figure 62 for the legend. The required level of capability for this test bed was 4.2 and that was easily met by the swarm as seen in Figure 68. Again, the gradually sloping parts of this plot are from the robots moving towards the target with the camera fixed on the target location. The sharp, non-smooth components are caused by robots losing and re-gaining line of sight on the target. At no time did any of the robots strike any obstacle or each other, and they were still able to deliver the appropriate amount of capability.

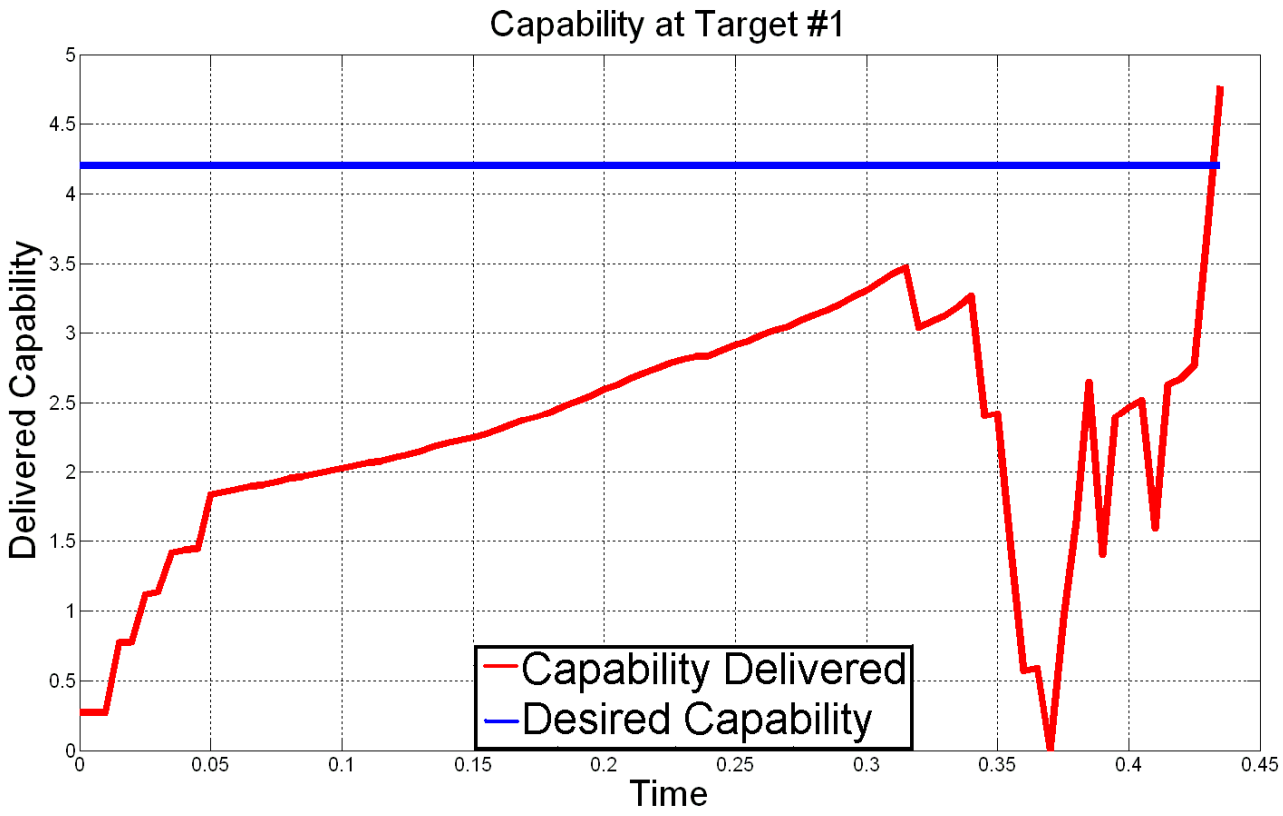

Figure 68: Test Bed #3 Delivered Capability

D. Test Bed #4

This test bed adds one final element to the controller. As in Test Bed #3, there are obstacles present and the swarm was initially set up in a spread formation. However, in this trial, each unit was "equipped" with a different type of virtual lens system, varying in focal length. The controller now had to account for the heterogeneous nature of the swarm. The virtual lenses for the swarm were 35 mm, 50 mm, 75 mm, 100 mm and 150 mm. Because of the varying focal length, each unit's optimal distance (for maximum capability) will be different from the others. The initial test bed setup is shown in Figure 69.

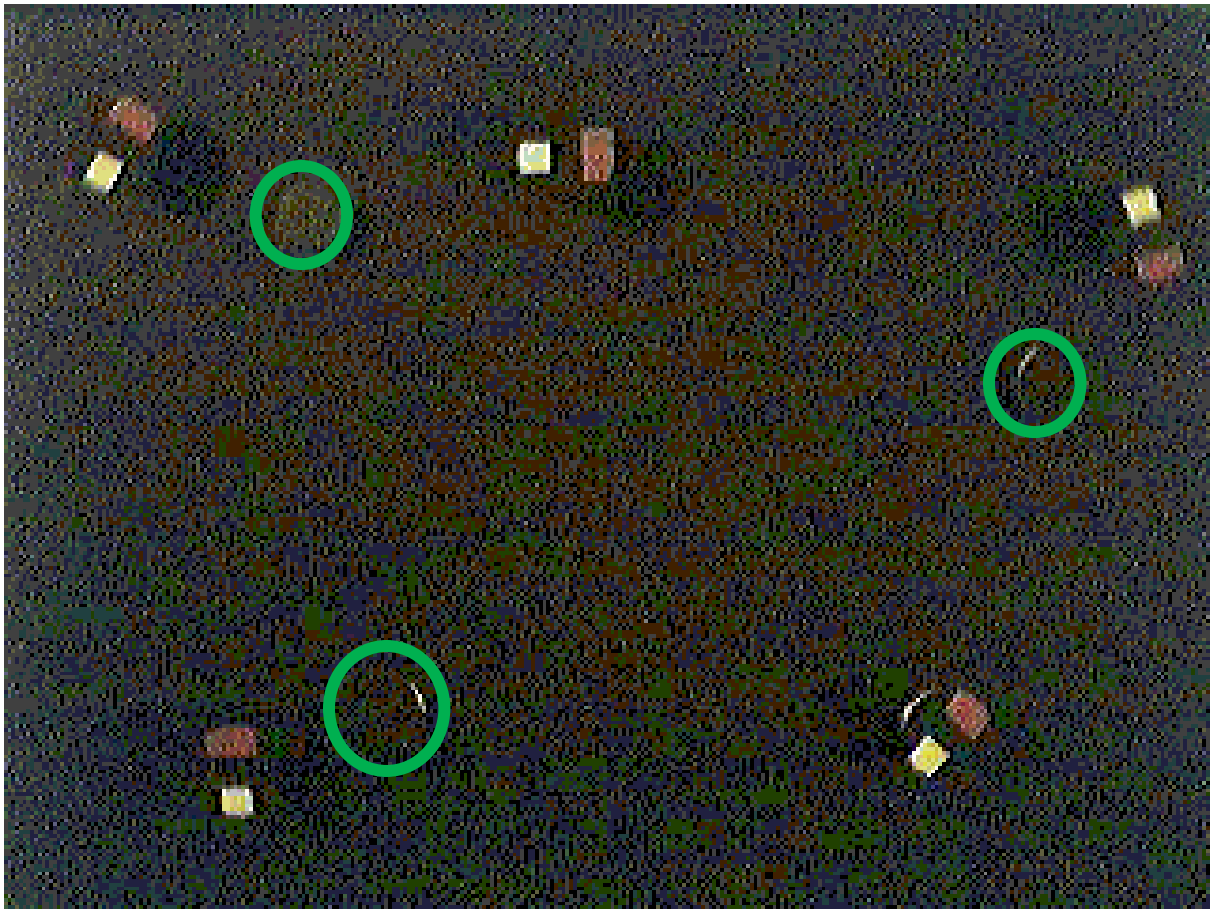

Figure 69: Camera View of Initial Set-up

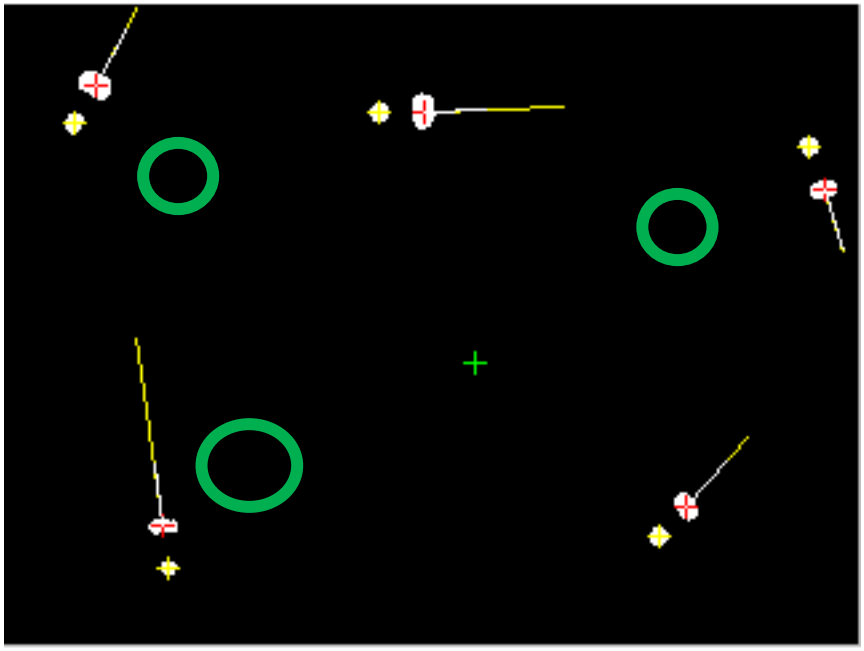

Figure 70: Binarized Computer Image of Initial Set-up

In Figure 70, one can see the different lengths in the yellow lines for each unit. This length corresponds to the optimal distance (maximal capability) for each lens. Each unit moved to an appropriate location to achieve desired capability, as seen in Figure 71

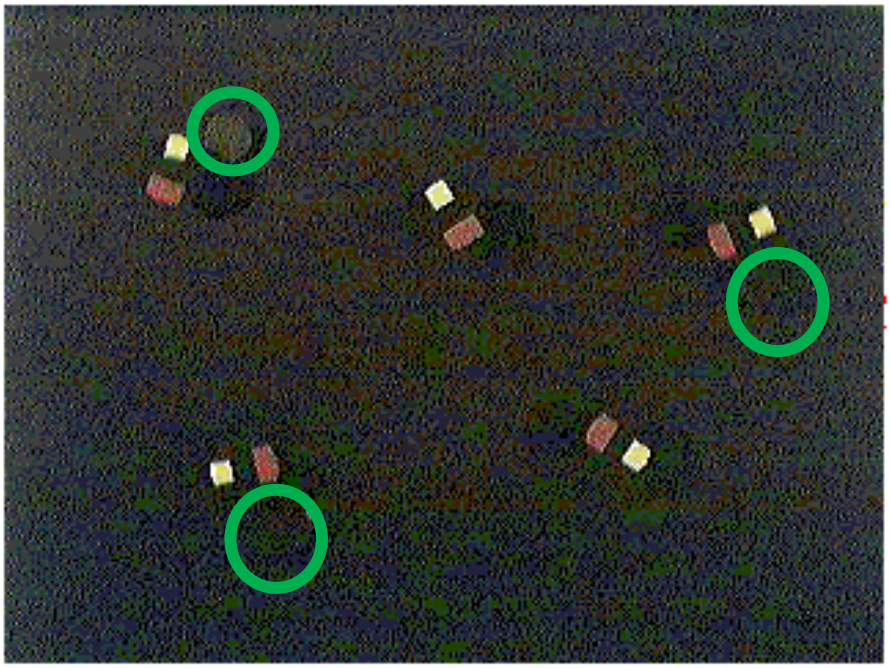

Figure 71: Camera View of Final Position

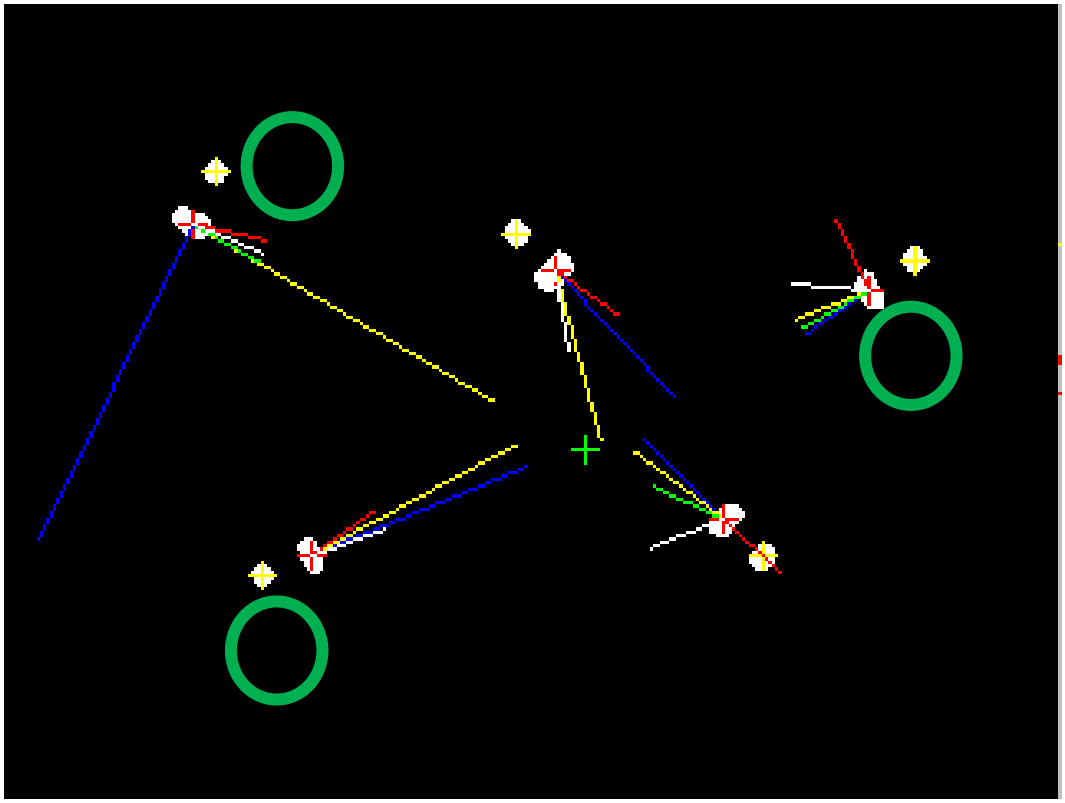

Figure 72: Binarized Computer Image of Final Position

As shown by the different camera indicators in Figure 72, each unit was positioned in a way that the capability met the desired level. While the robot on the far left was somewhat further away than the other units, it was in the process of driving towards the location when the rest of the swarm was already delivering sufficient capability to meet the required amount. Once again each unit was able to successfully avoid all obstacles in the environment and the overall trial run was very successful.

This test bed required a capability level of 4.2 delivered on target. This was easily and quickly met because the robots did not have to drive as far due to the larger lens systems. While this was perhaps the quickest solution out of all the test beds, there were some slight fluctuations with the capability level right before it met the requirement. This was due to the obstacle avoidance. Sometimes, in order to avoid an obstacle, the robot must turn so quickly that the

virtual camera system cannot react fast enough. However, this was quickly remedied, resulting in the spikes in capability as seen in Figure 73. Recall that this was *delivered* capability, not navigational capability. The plots for navigational capability are always smooth, as that concept does not require line of sight.

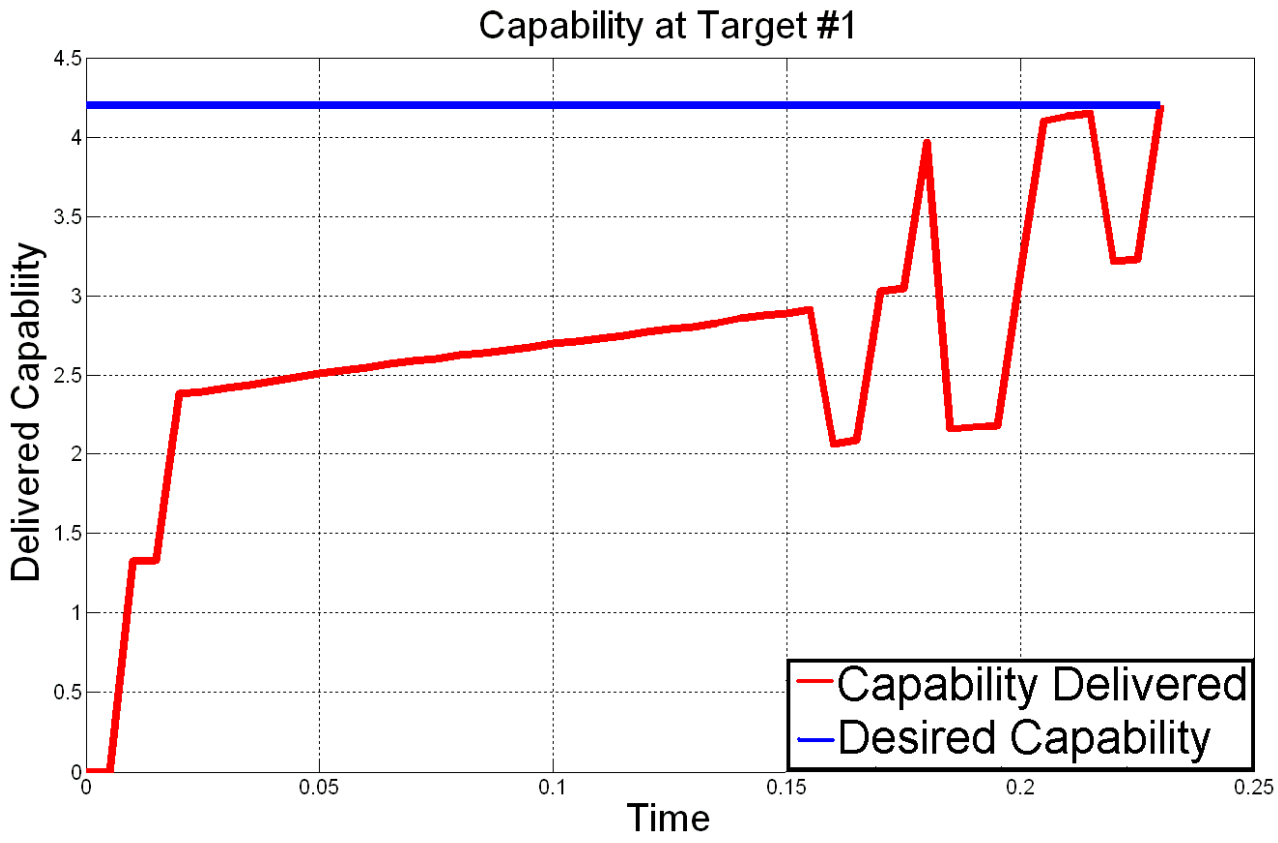

Figure 73: Test Bed #4 Delivered Capability

E. Test Bed #5

In this test bed, the environment changed and the swarm was again homogeneous (with a 35 mm virtual lens system). The main purpose of this trial was the test a different approach avenue by the robots to the target location. By having all of the robots approach from the same side and navigate a field of obstacles, a strong reliance on the robot avoidance system was put in

place. Since, all the units approached the target location from the same general direction, they naturally became converge toward another. Therefore, the controller had to generate motion commands to deliver the appropriate capability without robot-on-robot collisions. The initial positions can be seen in Figure 74 and Figure 75.

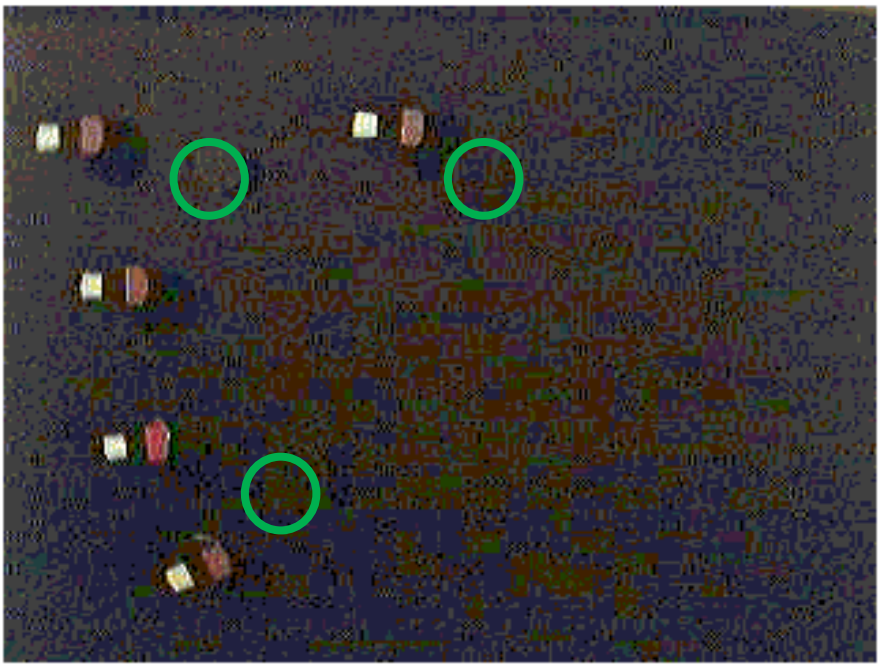

Figure 74: Camera View of Initial Set-up

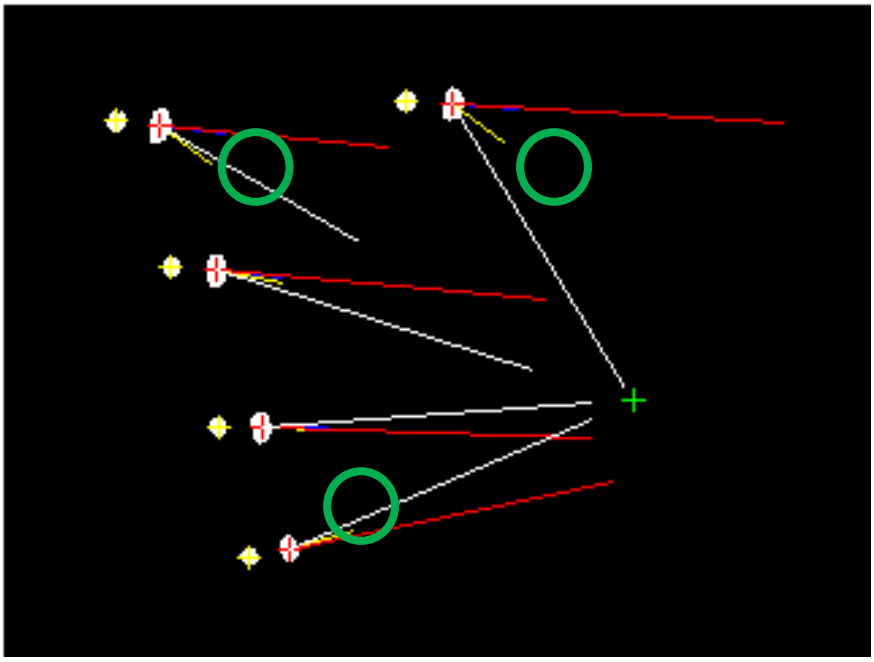

Figure 75: Binarized Computer Image of Initial Set-up

Again, the length of the red and white lines in Figure 75 correspond to the strength of the desired motion. A side note about the red repulsion vectors was that the length will always match the length of the white line regardless of what force was acting on them; however, when there was no force they would be displayed in the direction of the current robot heading, but would have no impact on desired motion. Since the robots were relatively far away and not delivering any capability, the attractive force was very strong. The yellow lines however remained the same set length depending on the focal length of the lens system.

The final positions of the system can be seen in Figure 76. While the units were bunched up at the final position, they never actually ran into each other during the final approach. At the point at which the image was taken, the desired capability was being delivered. If a higher capability was needed, the repulsive forces between the robots would have eventually forced them to move around the target location. The force vectors can be seen in Figure 77.

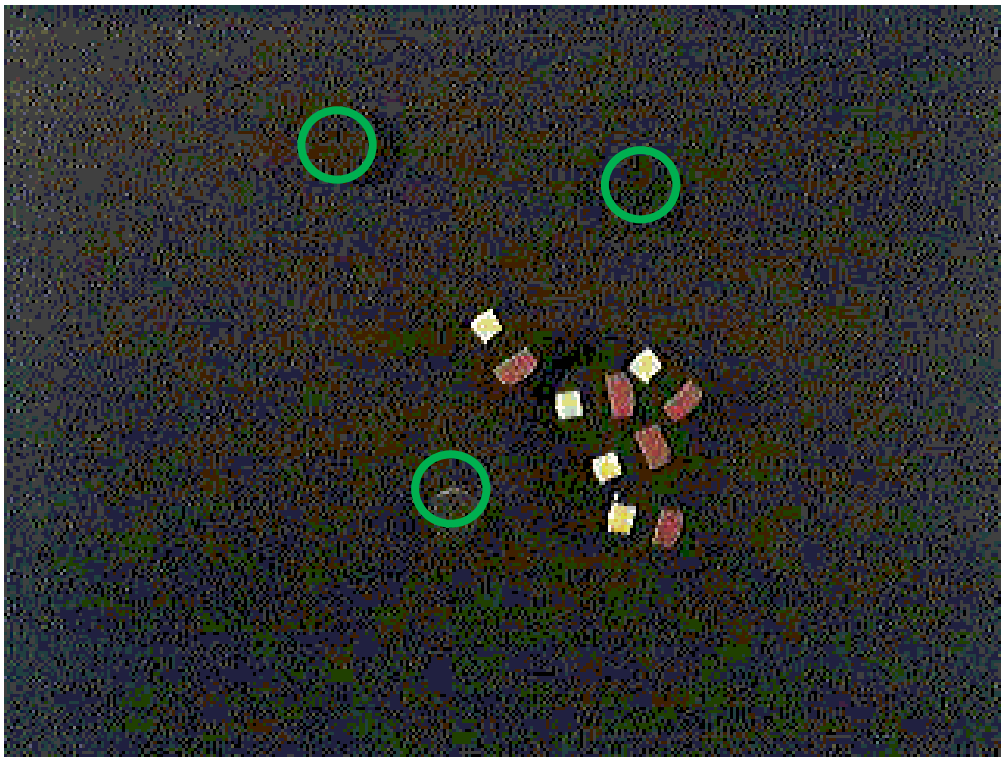

Figure 76: Camera View of Final Position

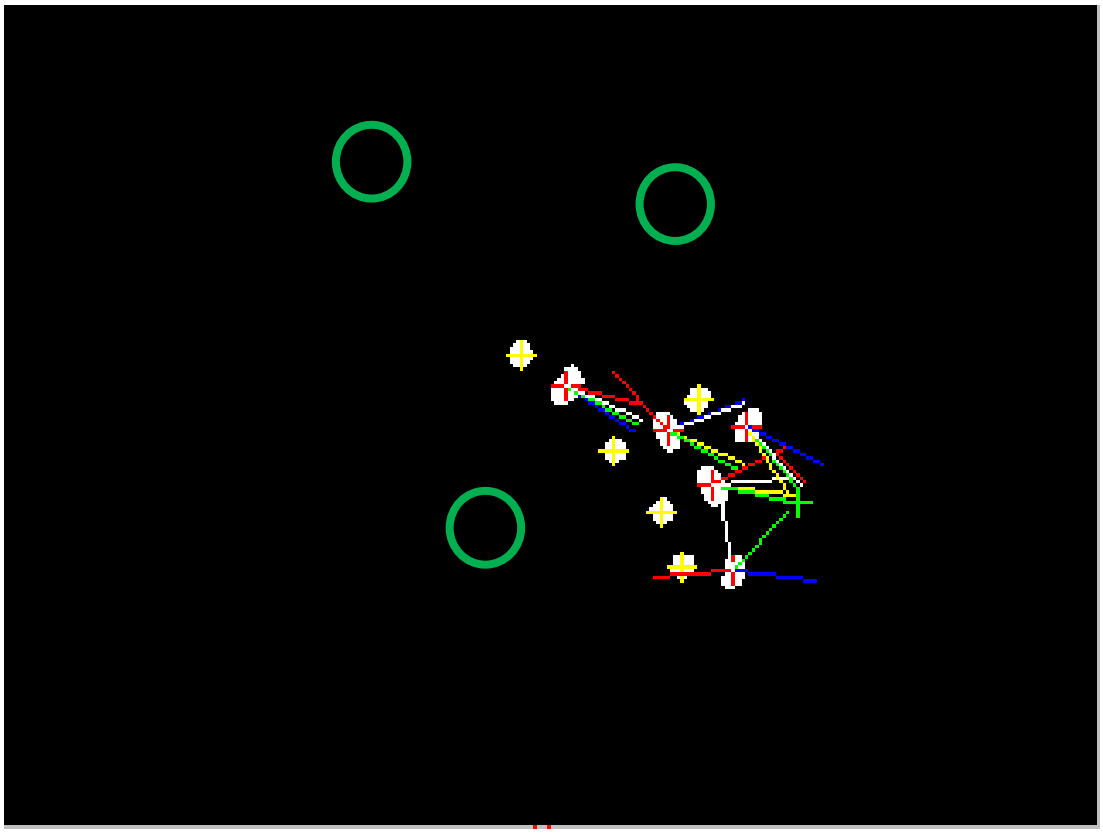

Figure 77: Binarized Computer Image of Final Position

The capability results from this trial were very similar to the others in the sense that there was a gradual increase in delivered capability while the robots were approaching the target followed by a series of spikes and drops as they adjusted position close to the target. Unlike the prior test bed where the robots were avoiding obstacles, in this trial they were avoiding the other robots as well, and the adjustments made in robot position to avoid collision sometimes causes the target to move out of the field of view of the camera. This trial also required a capability level of 4.2 to be delivered. The capability plot for this test bed can be seen in Figure 78.

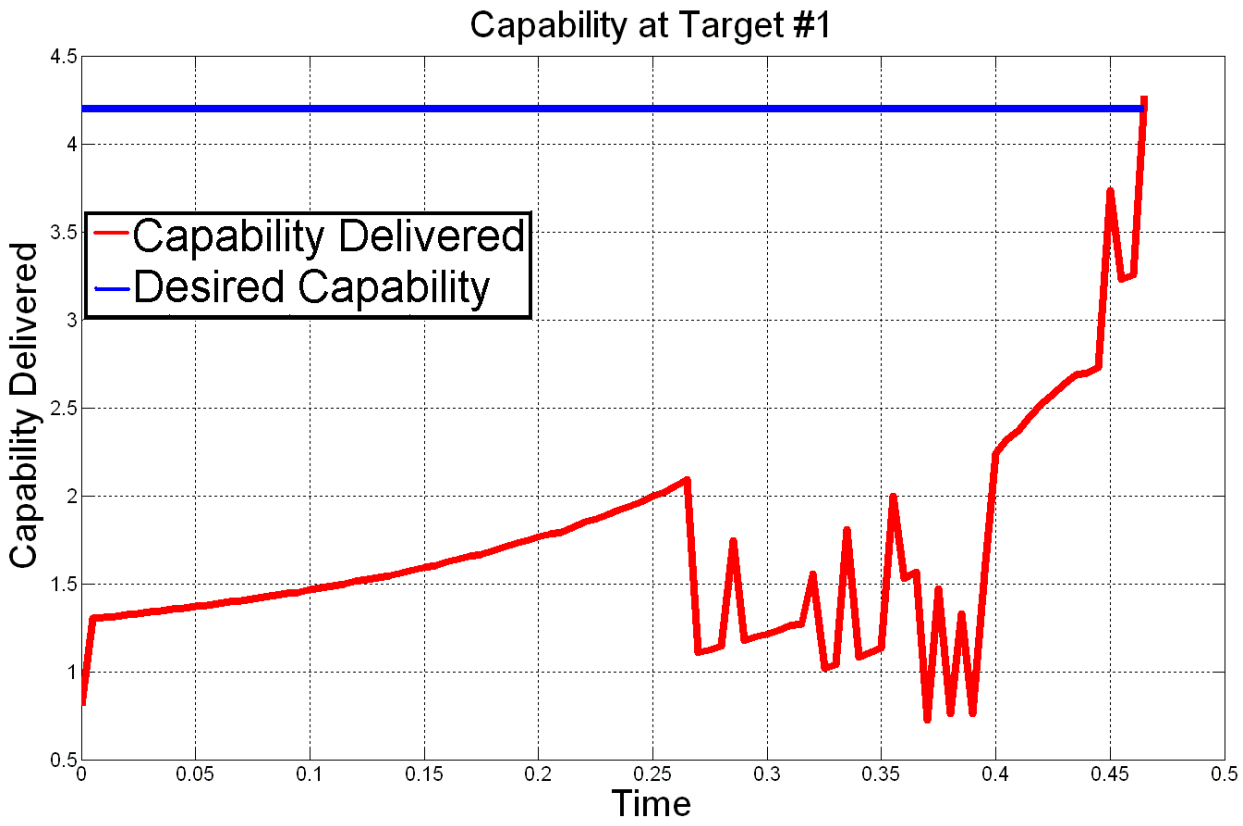

Figure 78: Test Bed #5 Delivered Capability

F. Test Bed #6

This final test bed followed the same principles as test bed #5 except the swarm was now heterogeneous. The initial set up for test bed #6 can be seen in Figure 79 and Figure 80 while the final positions of the swarm units can be seen in Figure 81 and Figure 82.

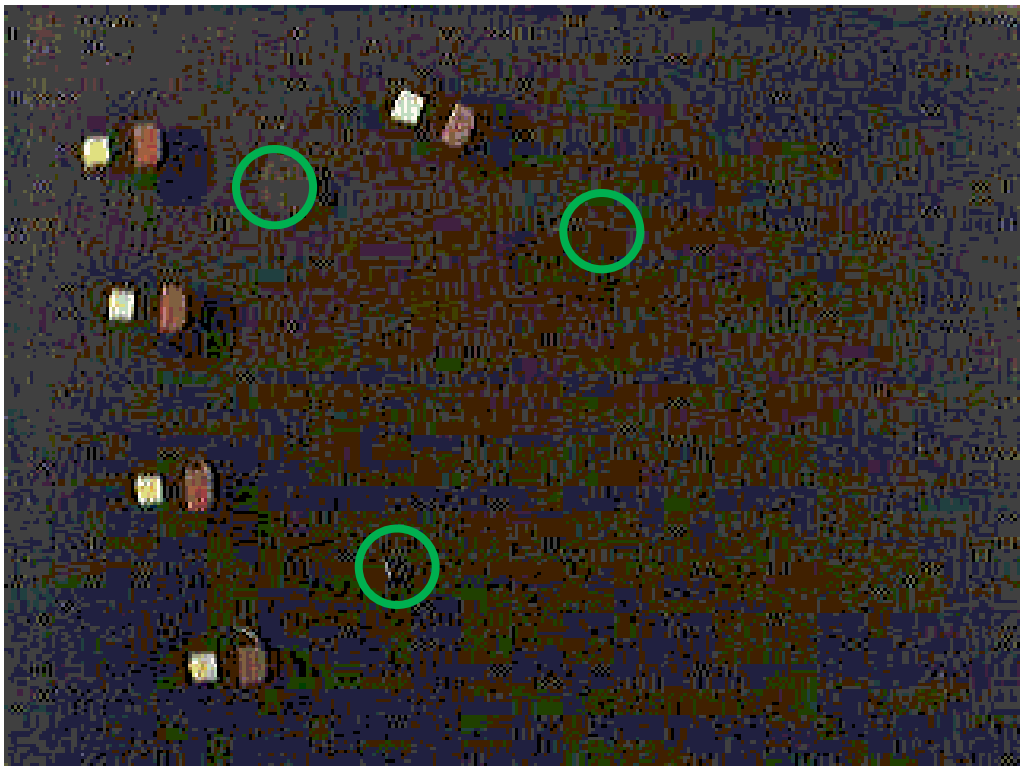

Figure 79: Camera View of Initial Set-up.

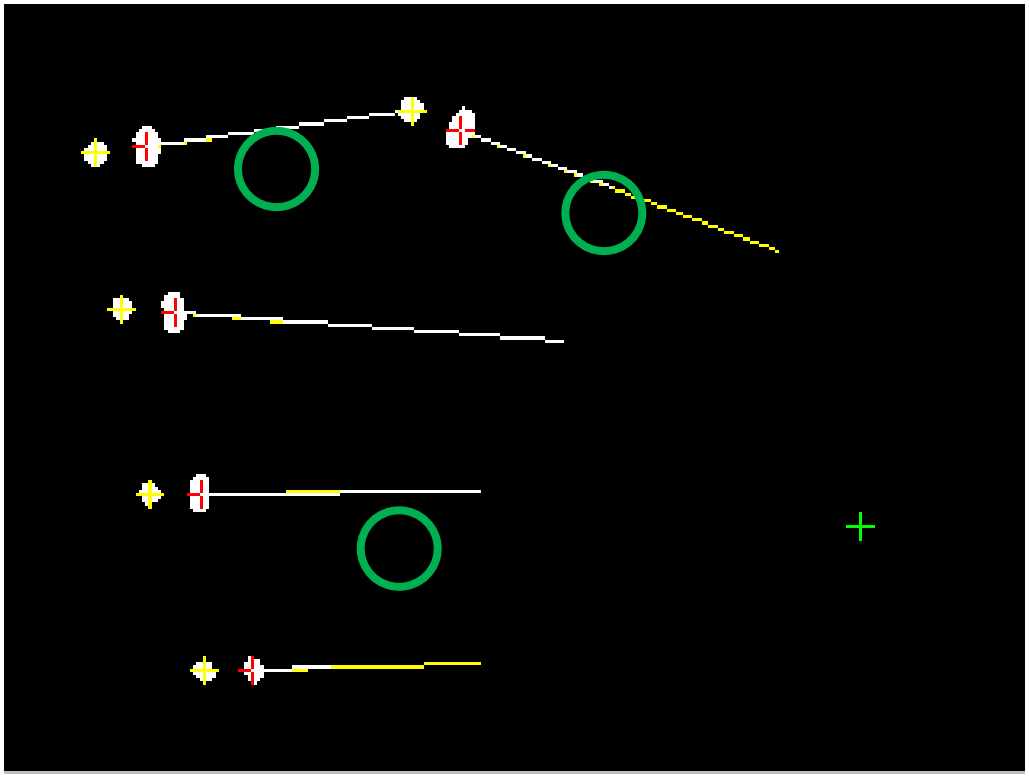

Figure 80: Binarized Computer Image of Initial Set-up.

Figure 81: Camera View of Final Position

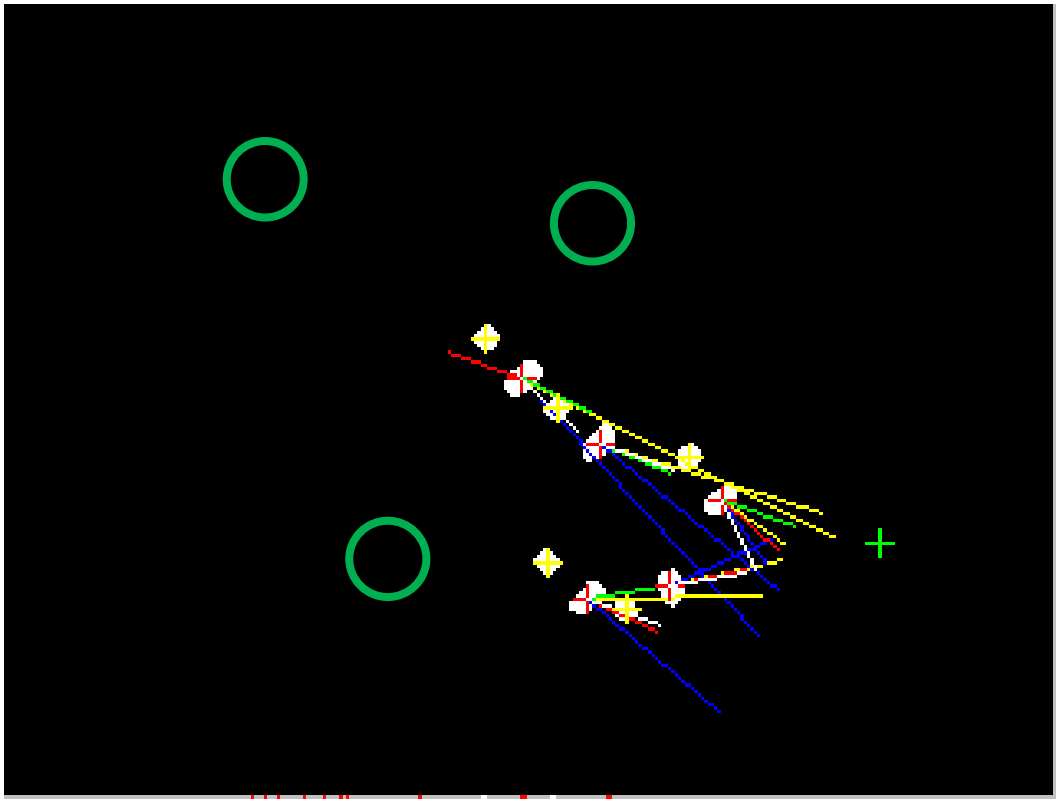

Figure 82: Binarized Computer Image of Final Position

For this last trial, the desired capability was once again set a level 4.2. The swarm was very successful at meeting this requirement in a timely manner while still avoiding all obstacles and each other. The actual delivered capability plot can be seen below in Figure 83.

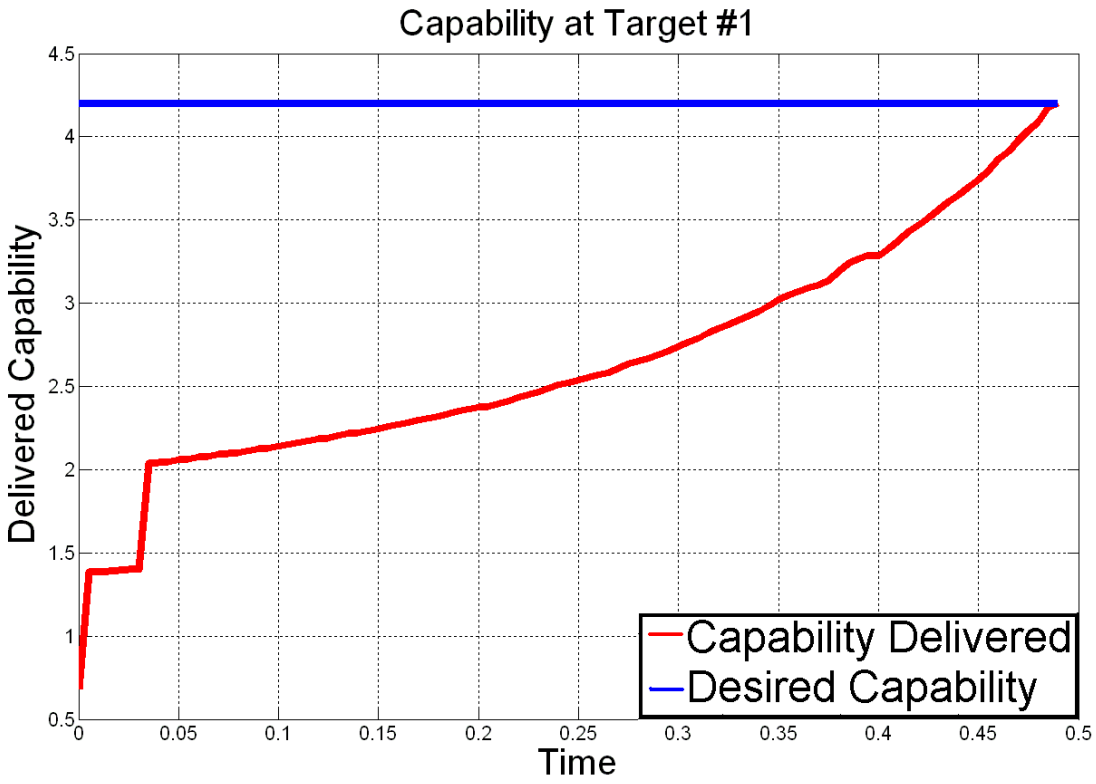

Figure 83: Test Bed #6 Delivered Capability

Overall, these six test beds have proven very successful and provide tremendous insight into this novel concept of swarm control. By successfully completing a variety of test beds that were built upon the initial 4 fundamentals, this research was able to prove the potential of this control concept.

10. Conclusion and Future Work

Throughout the course of this research, it was shown that a group of autonomous robots, equipped with a form of sensing capability, are able to navigate an unknown environment and position themselves in such a way that they are best able to carry out their mission. This was accomplished while also facing various secondary tasks such as obstacle avoidance, line of sight determination, ability to handle multiple objectives and the ability to control a heterogeneous swarm of units. Overall, the insight this projected has provided into the wide range of possibilities for swarm control, has created a very strong foundation to build upon for the advancement of capability based controllers. By proving the theory in the simulation phase, and then to validate it with hardware in a real environment, irrefutable evidence was provided that this theory is not only possible, but also desirable in many situations.

While this project was very successful, there is still tremendous opportunity for follow up research in the field of swarm robotics. The next step for this path would be to actually apply the developed methodologies outside of a lab with real field-operational robots equipped with GPS, actual camera systems, and improved range sensors for obstacle avoidance. The backbone of this project was to investigate the concepts and theories behind a capability based controller, and while the hardware implementation proved that the developed controller was successful, it would be interesting to see its performance in an operational environment.

Also, a different group of sensors could be implemented on the robotic platforms under this control scheme. In order to use this controller, all one would need to do is to be able to develop a capability function for whatever sensor they choose. There would be no other major modifications necessary. In the end, the realm of possibilities for swarm control is limitless, and this research has just scratched the surface. However, this research has opened new doors into a

revolutionary concept for swarm control, one that has the potential to have a tremendous impact on the world today and in the future.

References:

[1] Antonelli, G., Chiaverini, S., "**Kinematic Control of a Platoon of Autonomous Vehicles,**" *Proc. 2003 IEEE International Conference on Robotics and Automation*, pages 1464–1469, Taipei, Taiwan, Sept. 2003.

[2] Barnes, L.; Alvis, W.; Fields, M.; Valavanis, K.; Moreno, W., "**Heterogeneous Swarm Formation Control Using Bivariate Normal Functions to Generate Potential Fields"** *IEEE Workshop on Distributed Intelligent Systems: Collective Intelligence and Its Applications*, 2006

[3] Barnes, L.; Alvis, W.; Fields, M.; Valavanis, K.; Moreno, W., "**Swarm Formation Control with Potential Fields Formed by Bivariate Normal Functions"** *14th Mediterranean Conference on Control and Automation*, 2006

[4] Bishop, B.E., "**Control of Platoons of Nonholonomic Vehicles Using Redundant Manipulator Analogs"** *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*, 2005

[5] Bishop, B.E., "**Dynamics-Based Control of Robotic Swarms"** *Proceedings of the 2006 IEEE International Convention on Robotics and Automation*, 2006

[6] Bishop, B.E., "**On the use of Capability Functions for Cooperative Objective Achievement in Robot Swarms"** Sept, 2006

[7] Bishop, B.E., "**On the use of redundant manipulator techniques for control of platoons of cooperating robotic vehicles"** *IEEE Transactions on Systems, Man, and Cybernetics, Part A,* Sept. 2003.

[8] Dunbar, T., Esposito, J., "**Artificial Potential Field Controllers for Robust Communications in a Network of Swarm Robots"** *Proceedings of the Thirty-Seventh Southeastern Symposium on System Theory*, 2005.

[9] Goldbeck, J., "**Evolving optimal parameters for swarm control"** *NASA/DoD Conference on Evolvable Hardware,* 2002.

[10] J. P. Desai, J. Ostrowski, and V. Kumar, "**Controlling formations of multiple mobile robots,**" *Proceedings of the IEEE International Conference on Robotics Automation*, Leuven, Belgium, 1998, pp. 2864–2869.

[11] Kennedy, J., "**Bare-Bones Particle Swarms**" *Proceedings of the 2003 IEEE Swarm Intelligence Symposium*, 2003.

[12] Khepera II Mobile Robot User Manual

[13] Taylor, C., and Spletzer, J., "**Dynamic Sensor Planning and Control for Optimally Tracking Targets**" *GRASP Laboratory – University of Pennsylvania*

[14] V. Isler, S. Khanna, J. Spletzer, and C. Taylor, "**Target Tracking with Distributed Sensors: The Focus of Attention Problem**" *Proceedings of the IEEE International Conference on Intelligent Robots and Systems*, 2003.

[15] White, T.; Pagurek, B., "**Towards multi-swarm problem solving in networks**" *Proceedings of the International Conference on Multi-Agent Systems,* 1998.
[16] Y. Yang, O. Brock and R. A. Grupen, **"Exploiting Redundancy to Implement Multi-Objective Behavior**," *Proceedings of the 2003 IEEE International Conference on Robotics and Automation,* September 2003, pp. 3385 – 3390.
[17] Yong Chye Tan; Bishop, B.E., "**Combining Classical and Behavior-Based Control for Swarms of Cooperating Vehicles"** *Proceedings of the 2005 IEEE International Conference on Robotics and Autonomation*, 2005
[18] Yong Chye Tan; Bishop, B.E., "**Evaluation of robot swarm control methods for underwater mine countermeasures"** *Proceedings of the thirty-sixth southeastern symposium on Systems Theory*, 2004.

Appendices

Phase 1

Appendix A (Achieve_LOS.m)

```
function [RobotLOSX RobotLOSY]=Achieve_LOS(BotNumber, TgtNumber)
%Determine Potential Field to Allow Robot to See the Target

global Robot Target VertPerObst OBST NumTgts RobotRad

quad2=0;
quad3=0;
[Dist2Obst ObstBearing Closest_Obst DA BA]=Find_Closest_Obstacle(BotNumber);
% for Tgt=1:NumTgts
%     if Tgt==1
%         Dt=sqrt((Target(Tgt,1)-Robot(BotNumber,1))^2+(Target(Tgt,2)-Robot(BotNumber,2))^2)-
RobotRad;
%         CloseTgt=Tgt;
%     end
%     Dt2=sqrt((Target(Tgt,1)-Robot(BotNumber,1))^2+(Target(Tgt,2)-Robot(BotNumber,2))^2)-
RobotRad;
%     if Dt2<Dt
%         Dt=Dt2;
%         CloseTgt=Tgt;
%     end
% end
% if CloseTgt==TgtNumber
   [LOS
Closest_Obst_FOV]=Line_Of_Sight(Robot(BotNumber,1),Robot(BotNumber,2),Target(TgtNumber,1),T
arget(TgtNumber,2));
```

```
if LOS == 0        %If Can't See the Target
   LeftOrRight=Find_Shortest_Path(BotNumber,Closest_Obst_FOV,TgtNumber);
   for NumVert = 1:VertPerObst(Closest_Obst_FOV)    % For each object vertex
      VertX = OBST(2*Closest_Obst_FOV-1, NumVert);
      VertY = OBST(2*Closest_Obst_FOV, NumVert);
      VertAngle = atan2(VertY-Robot(BotNumber,2),VertX-Robot(BotNumber,1));

      if VertAngle > pi/2
         quad2 = 1;
      elseif VertAngle < -pi/2
         quad3 = 1;
      end

      if quad2==1 && quad3==1
         break
      end

      if NumVert==1
         MinAngle=VertAngle;
         MaxAngle=VertAngle;
      elseif VertAngle<MinAngle
         MinAngle=VertAngle;
      elseif VertAngle>MaxAngle
         MaxAngle=VertAngle;
      end
   end
   if quad2==1 && quad3==1     %Change Coordinates from -pi:pi to 0:2*pi
      for NumVert = 1:VertPerObst(Closest_Obst_FOV)    % For each object vertex
         VertX = OBST(2*Closest_Obst_FOV-1, NumVert);
         VertY = OBST(2*Closest_Obst_FOV, NumVert);
         VertAngle = atan2(VertY-Robot(BotNumber,2),VertX-Robot(BotNumber,1));

         if VertAngle < 0
            VertAngle = 2*pi + VertAngle;
         end
         if NumVert==1
            MinAngle=VertAngle;
            MaxAngle=VertAngle;
         elseif VertAngle<MinAngle
            MinAngle=VertAngle;
         elseif VertAngle>MaxAngle
            MaxAngle=VertAngle;

         end
      end
   end
   if LeftOrRight==1       %Moves Robot 15 degrees to the left of the left most vertex
      RobotLOSX=6*cos(MaxAngle+pi/6);
      RobotLOSY=6*sin(MaxAngle+pi/6);
   else                    %Moves Robot 15 degrees to the right of the left most vertex
```

```
        RobotLOSX=6*cos(MinAngle-pi/6);
        RobotLOSY=6*sin(MinAngle-pi/6);
      end
   %    RobotMove=atan2(RobotLOSY,RobotLOSX)
   %    Closest_Obst_FOV
   %    Closest_Obst

   else
      RobotLOSX=0;
      RobotLOSY=0;
   end
% else
%    RobotLOSX=0;
%    RobotLOSY=0;
% end
```

Appendix B (Capability.m)

```
function C = Capability()     %Returns The Jacobian and Capability Values per Robot
global  M Opt_Pix dPix NumBots NumTgts Robot Target

   %This for statement cycles through the number of robots and calculates
   %the capability being delivered to each of the target locations.
Cb=[];
Ct=[];
for TgtNumber=1:1:NumTgts
   for BotNumber=1:1:NumBots
      D(BotNumber)=sqrt((Robot(BotNumber,1)-Target(TgtNumber,1))^2+(Robot(BotNumber,2)-
Target(TgtNumber,2))^2);
      Tgt_Angle(BotNumber)=atan2(Target(TgtNumber,2)-Robot(BotNumber,2),Target(TgtNumber,1)-
Robot(BotNumber,1));

LOS=Line_Of_Sight(Robot(BotNumber,1),Robot(BotNumber,2),Target(TgtNumber,1),Target(TgtNumb
er,2));
      Diff_Angle=abs(Robot(BotNumber,3)-Tgt_Angle(BotNumber));
      while Diff_Angle>(2*pi)
         Diff_Angle=Diff_Angle-2*pi;
      end
      if Diff_Angle>pi
         Diff_Angle=2*pi-Diff_Angle;
      end
      if LOS==1 && Diff_Angle<(pi*.035/(8*Robot(BotNumber,5)))      %If the target is in the Field of
View
         if D<Robot(BotNumber,6)
            Cb(BotNumber)=sin((pi/2)*(D(BotNumber)/Robot(BotNumber,6)))^M*(10^(-
sin((Robot(BotNumber,3)-Tgt_Angle(BotNumber))/2)^4));
         else
```

```
            Cb(BotNumber)=sin((pi/2)*Robot(BotNumber,5)/(D(BotNumber)*dPix*Opt_Pix))*(10^(-
sin((Robot(BotNumber,3)-Tgt_Angle(BotNumber))/2)^4));
         end
      else
         Cb(BotNumber)=0;
      end
   end
   Ct=[Ct;Cb];
end
C=Ct;
```

Appendix C (Create_Environment.m)

```
global VertPerObst
global OBST
global Target
global Robot_Init

NumObst = input('How many obstacles do you wish to generate? '); %Determines Number of Obstacles
in Task Space
figure(1);
clf;
axis([0 15 0 15]);
hold on
OBST = [];              %Obstacle Matrix which Contains X and Y Coord. for all Vertices
VertPerObst = [];       %Length=NumObst. Each Data Point Tells Number of Vertices per Cooresponding
Obstacle

for i = 1:NumObst
   temp = ['Obstacle #', num2str(i), ':  Click on each vertex in CLOCKWISE order. Double-click final
vertex.'];
   disp(temp);
   input('Press ENTER when ready');
   [xo, yo] = getpts;
   [n, m] = size(OBST);
   xo = [xo; xo(1)];          %Last space closes obstacle.
   yo = [yo; yo(1)];
   fill(xo, yo, 'r');
   NumVert = length(xo);
   if (NumVert< 3)
      error('An obstacle must have at least 3 vertices');
   end
   VertPerObst(i) = NumVert - 1;
   if (m < NumVert)&&(m > 0)
```

```
      OBST = [OBST, zeros(n, NumVert - m)];
   end
   if (m > NumVert)
      xo = [xo; zeros(m-NumVert, 1)];
      yo = [yo; zeros(m-NumVert, 1)];
   end
   OBST = [OBST; xo'; yo'];
end


temp = 'Please Select Target Locations. (Double Click Final Target)';
disp(temp);
input('Press ENTER when ready');
[X,Y] = getpts;
Target=[X Y];
[n,m]=size(Target);
for a=1:n
   plot(Target(a,1),Target(a,2),'g+')
end


temp = 'Please Select Robot Starting Locations. (Double Click Final Robot)';
disp(temp);
input('Press ENTER when ready');
[X,Y] = getpts;
Robot_Init=[X Y];
[n,m]=size(Robot_Init);
Robot_Init=[Robot_Init zeros(n,1)];
for a=1:n
   plot(Robot_Init(a,1),Robot_Init(a,2),'b+')
end
```

Appendix D (Find_Closest_Obstacles.m)

```
function [Closest_Dist Bearing Closest_Obst DistArray BearingArray] =
Find_Closest_Obstacle(BotNumber)
%Returns the Distance and Bearing to the closest Obstacle

global OBST Robot VertPerObst RobotRad
DistArray=[];
BearingArray=[];
for NumObst = 1:length(VertPerObst)

   %**********************************
   % Locates Nearest Obstacle Vertex
   %**********************************
   Close_Vert=1;
```

```
   Closest_DistT=100000;
   for NumVert = 1:VertPerObst(NumObst)    % For each object vertex
      VertX = OBST(2*NumObst-1, NumVert);
      VertY = OBST(2*NumObst, NumVert);
      VertAngle = atan2(VertY-Robot(BotNumber,2),VertX-Robot(BotNumber,1));     %Calculate Angle
and Distance from Robot to Vertex
      VertDist = sqrt((VertX-Robot(BotNumber,1))^2+(VertY-Robot(BotNumber,2))^2)-RobotRad;

      if NumVert == 1 && NumObst==1 % First vertex- Set min and max
         Bearing = VertAngle;
         BearingT = VertAngle;
         Closest_Dist = VertDist;
         Closest_DistT = VertDist;
         Close_Vert=NumVert;
         Closest_Obst=NumObst;

      elseif NumVert == 1 && NumObst~=1
         BearingT = VertAngle;
         Closest_DistT = VertDist;
         Close_Vert=NumVert;
      else       % Find actual min values
         if VertDist < Closest_DistT
            Closest_DistT = VertDist;
            BearingT=VertAngle;
            Close_Vert=NumVert;
         end
      end
   end
   %*****************************************************
   %Determines if Vertex or Border Intersection is Closer
   %*****************************************************
   if Close_Vert==1 || Close_Vert==VertPerObst(NumObst)+1
      Other_Vert1=VertPerObst(NumObst);
      Other_Vert2=2;
   elseif Close_Vert==VertPerObst(NumObst);
      Other_Vert1=Close_Vert-1;
      Other_Vert2=1;
   else
      Other_Vert1=Close_Vert-1;
      Other_Vert2=Close_Vert+1;
   end
   x1 = OBST(2*NumObst-1, Close_Vert);
   x2 = OBST(2*NumObst-1, Other_Vert1);
   x3 = OBST(2*NumObst-1, Other_Vert2);
   y1 = OBST(2*NumObst, Close_Vert);
   y2 = OBST(2*NumObst, Other_Vert1);
   y3 = OBST(2*NumObst, Other_Vert2);
   a1=x2-x1;
   a2=x3-x1;
   if a1==0
      x2=x2+.0001;
```

```
    end
    if a2==0
       x3=x3+.0001;
    end
    a1=y2-y1;
    a2=y3-y1;
    if a1==0
       y2=y2+.0001;
    end
    if a2==0
       y3=y3+.0001;
    end
    VertSlope1 = (y2-y1)/(x2-x1);
    VertSlope2 = (y3-y1)/(x3-x1);
    VertYIntercept1 = y1 - x1*VertSlope1;
    VertYIntercept2 = y1 - x1*VertSlope2;
    RobotYIntercept1=Robot(BotNumber,2)-Robot(BotNumber,1)*(-1/VertSlope1);
    RobotYIntercept2=Robot(BotNumber,2)-Robot(BotNumber,1)*(-1/VertSlope2);

    %Find Intersection Locations

    [Xint1,Yint1] = intersection(-1/VertSlope1,RobotYIntercept1,VertSlope1,VertYIntercept1);
    [Xint2,Yint2] = intersection(-1/VertSlope2,RobotYIntercept2,VertSlope2,VertYIntercept2);

    %If intersection is on the Obstacle Border, than that intersection
    %point is the closest point to the robot.

    if Xint1 > min(x1,x2) && Xint1 < max(x1,x2) && Yint1 > min(y1,y2) && Yint1 < max(y1,y2)
       Closest_DistT=sqrt((Robot(BotNumber,1)-Xint1)^2+(Robot(BotNumber,2)-Yint1)^2)-RobotRad;
       BearingT=atan2(Yint1-Robot(BotNumber,2),Xint1-Robot(BotNumber,1));
    elseif Xint2 > min(x1,x3) && Xint2 < max(x1,x3) && Yint2 > min(y1,y3) && Yint2 < max(y1,y3)
       Closest_DistT=sqrt((Robot(BotNumber,1)-Xint2)^2+(Robot(BotNumber,2)-Yint2)^2)-RobotRad;
       BearingT=atan2(Yint2-Robot(BotNumber,2),Xint2-Robot(BotNumber,1));
    end
    if Closest_DistT < Closest_Dist
       Closest_Dist=Closest_DistT;
       Bearing=BearingT;
       Closest_Obst=NumObst;
    end
    DistArray=[DistArray Closest_DistT];
    BearingArray=[BearingArray BearingT];
end
```

Appendix E (Find_Shortest_Path.m)

```
function LeftOrRight = Find_Shortest_Path(BotNumber,Closest_Obst,TgtNumber)
```

```
%Determines whether robot should move to the left or right of an obstacle
%to find quickest path to achieve Line of Sight

global OBST Robot VertPerObst Target

quad2=0;
quad3=0;

for NumVert = 1:VertPerObst(Closest_Obst)    % For each object vertex
      VertX = OBST(2*Closest_Obst-1, NumVert);
      VertY = OBST(2*Closest_Obst, NumVert);
      VertAngle = atan2(VertY-Target(TgtNumber,2),VertX-Target(TgtNumber,1));
      Tgt_Angle=atan2(Robot(BotNumber,2)-Target(TgtNumber,2),Robot(BotNumber,1)-
Target(TgtNumber,1));

      if VertAngle > pi/2
         quad2 = 1;
      elseif VertAngle < -pi/2
         quad3 = 1;
      end

      if quad2==1 && quad3==1
         break
      end

      if NumVert==1
         MinAngle=VertAngle;
         MaxAngle=VertAngle;
      elseif VertAngle<MinAngle
         MinAngle=VertAngle;
      elseif VertAngle>MaxAngle
         MaxAngle=VertAngle;
      end
end
if quad2==1 && quad3==1
   for NumVert = 1:VertPerObst(Closest_Obst)    % For each object vertex
      VertX = OBST(2*Closest_Obst-1, NumVert);
      VertY = OBST(2*Closest_Obst, NumVert);
      VertAngle = atan2(VertY-Target(TgtNumber,2),VertX-Target(TgtNumber,1));
      Tgt_Angle=atan2(Robot(BotNumber,2)-Target(TgtNumber,2),Robot(BotNumber,1)-
Target(TgtNumber,1));

      if VertAngle < 0
         VertAngle = 2*pi + VertAngle;
      end
      if Tgt_Angle < 0
         Tgt_Angle = 2*pi + Tgt_Angle;
      end
      if NumVert==1
         MinAngle=VertAngle;
         MaxAngle=VertAngle;
```

```
      elseif VertAngle<MinAngle
         MinAngle=VertAngle;
      elseif VertAngle>MaxAngle
         MaxAngle=VertAngle;

      end
   end
end

DiffAng1=abs(Tgt_Angle-MinAngle);
DiffAng2=abs(Tgt_Angle-MaxAngle);
if DiffAng1<DiffAng2
   LeftOrRight=1;         %Move Left
else
   LeftOrRight=2;         %Move Right
end
```

Appendix F (Focal_Length_Comparison.m)

```
f1=.035;
f2=.05;
f3=.075;
f4=.100;
f5=.125;
f6=.150;
dPix=.0000856;
d=.5:.01:10;
Opt_Pix(1:length(d))=400;
Pixel1=f1./(d.*dPix);
Pixel2=f2./(d.*dPix);
Pixel3=f3./(d.*dPix);
Pixel4=f4./(d.*dPix);
Pixel5=f5./(d.*dPix);
Pixel6=f6./(d.*dPix);
plot(d,Pixel1,'b',d,Pixel2,'g',d,Pixel3,'y',d,Pixel4,'c',d,Pixel5,'m',d,...
   Pixel6,'k',d,Opt_Pix,'r')
grid on;
xlabel('Distance from Camera Lens (m)')
ylabel('Pixels per Meter of Target Area')
title('Comparison of Pixel Resolution for 35mm - 150mm Lenses')
```

Appendix G (Init_Capability_Func.m)

```
G=1;
A=2;
M=5;
dPix=.0000856;
f=.1;
Opt_Pix=400;
k=f/(Opt_Pix*dPix);

%*********************************************************
%   2D PLOT OF INDIVIDUAL CAPABILITY FUNCTION
%*********************************************************
figure(1);
F=@(d)G.*(sin((pi/2)*(d./k)).^M);

F1=@(d)G.*sin((pi/2)*(f/(d*dPix)/Opt_Pix));

fplot(F,[0 k]); hold on; fplot(F1,[k 8])
axis([0 8 0 1.1]);
%*********************************************************************
%    3D POLAR PLOT OF INDIVIDUAL CAPABILITY FUNCTIONS W/ DIRECTION
%*********************************************************************
figure(2);
title('Navigational Capability')
Heading=90*pi/180;
[th,R] = meshgrid((0:1:360)*pi/180,0:.01:k);
[X,Y] = pol2cart(th,R);
F=(G.*(sin((pi/2)*(R./k)).^M)).*(10.^(-sin(abs(Heading-th)/2).^4));
surf(X,Y,F)
hold on
[th,R] = meshgrid((0:1:360)*pi/180,k:.01:8);
[X,Y] = pol2cart(th,R);
F=G.*sin((pi/2)*(f./(R.*dPix)/Opt_Pix)).*(10.^(-sin(abs(Heading-th)/2).^4));
surf(X,Y,F)
pause(.0001);
hold off


figure(3);
title('Deliverable Capability')
Heading=90*pi/180;
ZOOM=10;
[th,R] = meshgrid((-247:1:67)*pi/180,0:.01:8);
[X,Y] = pol2cart(th,R);
F=0.*X.*Y;
surf(X,Y,F)
hold on
pause(.0001);
```

```
[th,R] = meshgrid((67:1:113)*pi/180,0:.01:k);
[X,Y] = pol2cart(th,R);
F=(G.*(sin((pi/2)*(R./k)).^M)).*(10.^(-sin(abs(Heading-th)/2).^4));
surf(X,Y,F)
hold on
[th,R] = meshgrid((67:1:113)*pi/180,k:.01:8);
[X,Y] = pol2cart(th,R);
F=G.*sin((pi/2)*(f./(R.*dPix)/Opt_Pix)).*(10.^(-sin(abs(Heading-th)/2).^4));
surf(X,Y,F)
pause(.0001);
hold off
```

Appendix H (Init_Controller.m)

```
global dt M Opt_Pix dPix NumBots NumTgts Robot Target OBST VertPerObst
global RobotRad PosHistory DistArray Robot_Init
temp=input('Would you like to use a new environment? (y=1 or n=2)');
if temp==1
   clc;
   Create_Environment
elseif temp==2
   clc;
   Print_Environment
else
   while temp~=1 && temp~=2
      temp=input('Invalid Key Pressed: Would you like to use a new environment? (y=1 or n=2)');
      if temp==1
         clc;clear;
         Create_Environment
      elseif temp==2
         Print_Environment
      end
   end
end

Robot=Robot_Init;

Robot(1,5)=.035;
Robot(2,5)=.035;
Robot(3,5)=.035;
Robot(4,5)=.035;

Robot(1,7)=30;
```

```
Robot(2,7)=30;
Robot(3,7)=30;
Robot(4,7)=30;

RobotRad=.1;
dt=.005;
M=5;
MIN_SPEED=8;
dPix=.0000856;          %Pixel width in m in the image frame.

Opt_Pix=400;            %Optimal Number of Pixels per Target Area
[NumBots a]=size(Robot);              %Number of Robots in Swarm
[NumTgts b]=size(Target);             %Number of Targets in System
for a=1:NumBots
   k=Robot(a,5)/(Opt_Pix*dPix); %Distance to Achieve Opt_Pix
   Robot(a,6)=k;
end

                  %Robot Matrix Robot=[ X1 Y1 Theta1 DesiredHeading1 FocalLength1 k1 MaxSpeed1]
                  %                    X2 Y2 Theta2 DesiredHeading2 FocalLength2 k2 MaxSpeed2]
                  %                    ...]
                  %Target Matrix Target=[X1 Y1
                  %                      X2 Y2
                  %                      ... ]
                  %Obstacle Matrix OBST=[Ax1 Ax2 Ax3...
                  %                      Ay1 Ay2 Ay3...
                  %                      Bx1 Bx2 Bx3...
                  %                      By1 By2 By3...]
kp=1;
kt=5;             %Proportional Gain
Cd=[3.95];   %***TEMPORARY*** (just makes sure each bot achieves max cap. delivery)
NotDone=0;
CHist=[];
Alert='MAX REACHED';
Alert2='MIN REACHED';

PosHistory=[];
Time=0;
while NotDone==0
   PHT=[];
   for B=1:1:NumBots
      PHT=[PHT;Robot(B,1);Robot(B,2)];
   end
   PosHistory=[PosHistory PHT];

   J=Jacobian();         %Returns a Jacobian Matrix for the system
   C=Capability();       %Returns the capability for each individual robot
   pJ=J'*((J*J')^(-1));            %Pseudo-Inverse for entire system
   CTotal=C(:,1);
   for i=2:1:NumBots
      CTotal=CTotal+C(:,i);                    %Sums all of the individual capabilities
```

```
    end
    CHist=[CHist CTotal];

    for z=1:length(Cd)
        if CTotal(z)>=Cd(z)
            NotDone=NotDone+1;
        end
    end
    if NotDone==length(Cd)
        break;
    end
    NotDone=0;
    TaskVar=Task_Space_Controller();
    a=length(J);
    I=eye(a);



    qDot=pJ*(kp*(Cd-CTotal))+kt*(I-pJ*J)*TaskVar;          %Desired Changes in X, Y AND Theta


    for a=1:NumBots
        DesiredHeading=atan2(qDot(3*a-1),qDot(3*a-2));
        Speed=sqrt(qDot(3*a-1)^2+qDot(3*a-2)^2);
        Robot(a,4)=DesiredHeading;
        if Speed>Robot(a,7)
            qDot(3*a-2)=Robot(a,7)*cos(DesiredHeading);
            qDot(3*a-1)=Robot(a,7)*sin(DesiredHeading);
        elseif Speed<MIN_SPEED
            qDot(3*a-2)=MIN_SPEED*cos(DesiredHeading);
            qDot(3*a-1)=MIN_SPEED*sin(DesiredHeading);
        end
    end
    %*****HARD CODE MUST BE CHANGED DEPENDING ON NUMBER OF ROBOTS******
    qDot2=[qDot(1:3)'; qDot(4:6)'; qDot(7:9)'; qDot(10:12)'];   %Changes qDot from a 12x1 Matrix to a
4x3 matrix corresponding to Robot



    %Adjust Robot X,Y and Theta Values Depending on qDot Value
    for BotNumber=1:1:NumBots
        Robot(BotNumber,1)=Robot(BotNumber,1)+qDot2(BotNumber,1)*dt;
        Robot(BotNumber,2)=Robot(BotNumber,2)+qDot2(BotNumber,2)*dt;
        Robot(BotNumber,3)=Robot(BotNumber,3)+3*qDot2(BotNumber,3)*dt;
    end

    %Plot Robot Locations with Orientation
    figure(1);
    for TgtNumber=1:1:NumTgts
        for BotNumber=1:1:NumBots
            axis([0 15 0 15]);
```

```
        hold on;

plot(Robot(BotNumber,1),Robot(BotNumber,2),'bo',Target(TgtNumber,1),Target(TgtNumber,2),'g+',[Ro
bot(BotNumber,1),...

Robot(BotNumber,1)+Robot(BotNumber,6)*cos(Robot(BotNumber,3))],[Robot(BotNumber,2),Robot(Bo
tNumber,2)+Robot(BotNumber,6)*sin(Robot(BotNumber,3))],'y');%,...
%
[Robot(BotNumber,1),Robot(BotNumber,1)+Robot(BotNumber,6)*cos(Robot(BotNumber,4))],[Robot(B
otNumber,2),Robot(BotNumber,2)+Robot(BotNumber,6)*sin(Robot(BotNumber,4))],'r');

      end
   end
   figure(2);
   hold off;
   axis([0 3 0 4])
   bar([CTotal Cd],'group')
   hold on;

   Time=Time+dt;

end

%Plot Robot Locations with Orientation
   figure(1);
   for TgtNumber=1:1:NumTgts
      for BotNumber=1:1:NumBots
         axis([0 15 0 15]);
         hold on;

plot(Robot(BotNumber,1),Robot(BotNumber,2),'bo',Target(TgtNumber,1),Target(TgtNumber,2),'g+',[Ro
bot(BotNumber,1),...

Robot(BotNumber,1)+Robot(BotNumber,6)*cos(Robot(BotNumber,3))],[Robot(BotNumber,2),Robot(Bo
tNumber,2)+Robot(BotNumber,6)*sin(Robot(BotNumber,3))],'y');%,...
%
[Robot(BotNumber,1),Robot(BotNumber,1)+Robot(BotNumber,6)*cos(Robot(BotNumber,4))],[Robot(B
otNumber,2),Robot(BotNumber,2)+Robot(BotNumber,6)*sin(Robot(BotNumber,4))],'r');

      end
   end
   figure(2);
   hold off;
   axis([0 3 0 4])
   bar([CTotal Cd],'group')
   hold on;

disp('DONE')

T=0:dt:dt*(length(CHist)-1);
n=length(T);
```

```
CDisp=[];
for TgtNumber=1:1:NumTgts
   for a=1:1:n
      CDisp(a)=Cd(TgtNumber);
   end
      figure(TgtNumber+2)
      plot(T,CHist(TgtNumber,:),'r',T,CDisp,'b');
      title(['Capability at Target #', num2str(TgtNumber)]);
      hold on;
end

figure(NumTgts+3);
axis([0 15 0 15]);
hold on
for i = 1:NumObst
   xo = OBST(2*i-1,1:VertPerObst(i));          %Last space closes obstacle.
   yo = OBST(2*i,1:VertPerObst(i));
   fill(xo, yo, 'r');

end
[n,m]=size(Target);
for a=1:n
   plot(Target(a,1),Target(a,2),'g+')
end
for BotNumber=1:1:NumBots

plot(Robot(BotNumber,1),Robot(BotNumber,2),'bo',Target(TgtNumber,1),Target(TgtNumber,2),'g+',[Ro
bot(BotNumber,1),...

Robot(BotNumber,1)+Robot(BotNumber,6)*cos(Robot(BotNumber,3))],[Robot(BotNumber,2),Robot(Bo
tNumber,2)+Robot(BotNumber,6)*sin(Robot(BotNumber,3))],'y');%,...
%
[Robot(BotNumber,1),Robot(BotNumber,1)+Robot(BotNumber,6)*cos(Robot(BotNumber,4))],[Robot(B
otNumber,2),Robot(BotNumber,2)+Robot(BotNumber,6)*sin(Robot(BotNumber,4))],'r');
   hold on;
end
```

Appendix I (intersection.m)

```
function [xint,yint] = intersection (slope1,y_inter1,slope2,y_inter2)

xint = (y_inter1-y_inter2)/(slope2-slope1);
yint = (-slope1*y_inter2+y_inter1*slope2)/(slope2-slope1);
```

Appendix J (Jacobian.m)

```
function J=Jacobian()     %Returns The Jacobian and Capability Values per Robot
global  M f Opt_Pix dPix NumBots NumTgts Robot Target

   %This for statement cycles through the number of robots and calculates
   %the partial derivative for movement in the X and Y direction as well
   %as a change in Theta.  These are then compiled into 1 Jacobian for
   %the entire system.
Jb=[];
Jt=[];
for TgtNumber=1:1:NumTgts
   for BotNumber=1:1:NumBots
      D(BotNumber)=sqrt((Robot(BotNumber,1)-Target(TgtNumber,1))^2+(Robot(BotNumber,2)-
Target(TgtNumber,2))^2);
      Tgt_Angle(BotNumber)=atan2(Target(TgtNumber,2)-Robot(BotNumber,2),Target(TgtNumber,1)-
Robot(BotNumber,1));
      if D<Robot(BotNumber,6)
         dCdX=M/4*sin(1/2*pi*D(BotNumber)/Robot(BotNumber,6))^(M-1)*(10^(-
sin((Robot(BotNumber,3)-Tgt_Angle(BotNumber))/2)^4))*...

cos(1/2*pi*D(BotNumber)/Robot(BotNumber,6))*pi/D(BotNumber)/Robot(BotNumber,6)*2*(Robot(Bo
tNumber,1)-Target(TgtNumber,1));
         dCdY=M/4*sin(1/2*pi*D(BotNumber)/Robot(BotNumber,6))^(M-1)*(10^(-
sin((Robot(BotNumber,3)-Tgt_Angle(BotNumber))/2)^4))*...

cos(1/2*pi*D(BotNumber)/Robot(BotNumber,6))*pi/D(BotNumber)/Robot(BotNumber,6)*2*(Robot(Bo
tNumber,2)-Target(TgtNumber,2));
         dCdTheta=-2*(sin(pi*D(BotNumber)/(2*Robot(BotNumber,6)))^M)*cos((Robot(BotNumber,3)-
Tgt_Angle(BotNumber))/2)*...
            (sin((Robot(BotNumber,3)-Tgt_Angle(BotNumber))/2)^3)*log(10)*10^(-
sin((Robot(BotNumber,3)-Tgt_Angle(BotNumber))/2)^4);
      else
         dCdX=2*(Robot(BotNumber,1)-Target(TgtNumber,1))*(-
pi/4*Robot(BotNumber,5)/(dPix*Opt_Pix)*D(BotNumber)^(-
3))*cos(pi/2*Robot(BotNumber,5)/(D(BotNumber)*...
            dPix*Opt_Pix))*10^(-(sin((Robot(BotNumber,3)-Tgt_Angle(BotNumber))/2)^4));
         dCdY=2*(Robot(BotNumber,2)-Target(TgtNumber,2))*(-
pi/4*Robot(BotNumber,5)/(dPix*Opt_Pix)*D(BotNumber)^(-
3))*cos(pi/2*Robot(BotNumber,5)/(D(BotNumber)*...
            dPix*Opt_Pix))*10^(-(sin((Robot(BotNumber,3)-Tgt_Angle(BotNumber))/2)^4));
         dCdTheta= -
2*sin((pi/2)*Robot(BotNumber,5)/(D(BotNumber)*dPix*Opt_Pix))*cos((Robot(BotNumber,3)-
Tgt_Angle(BotNumber))/2)*...
            (sin((Robot(BotNumber,3)-Tgt_Angle(BotNumber))/2)^3)*log(10)*10^(-
sin((Robot(BotNumber,3)-Tgt_Angle(BotNumber))/2)^4);
      end
      Jb=[Jb dCdX dCdY dCdTheta];
   end
```

```
   Jt=[Jt;Jb];
   Jb=[];
end
J = Jt;
```

Appendix K (Line_Of_Sight.m)

```
function [LOS Closest_Obst_FOV] = Line_Of_Sight (Rx,Ry,TargX,TargY)
%This function determines if Line of Sight Capability is Possible.

global VertPerObst
global OBST
global xint yint

LOS = 1;

Tgt_Angle = atan2(TargY-Ry,TargX-Rx);  %Angle between current position and Desired
Dist = sqrt((TargX-Rx)^2+(TargY-Ry)^2);    % Distance between desired and current
Slope2Tgt = (TargY-Ry)/(TargX-Rx);     % Slope of desired-current pos line
TgtYIntercept = Ry - Rx*Slope2Tgt;    % y intercept of desired-current pos line
Closest_Obst_FOV=0;
CO_MD=1000000;
for NumObst = 1:length(VertPerObst)   % For each object
   Quad2 = 0;  % flag for quadrant 2 object corner
   Quad3 = 0;  % flag for quadrant 3 object corner

   % Determine min and max distance to obstacle vertices
   % as well was the angles to the outermost left and right vertices

   for NumVert = 1:VertPerObst(NumObst)    % For each object vertex
      x1 = OBST(2*NumObst-1, NumVert);
      y1 = OBST(2*NumObst, NumVert);

      VertAngle = atan2(y1-Ry,x1-Rx);     %Calculate Angle and Distance from Robot to Vertex
      VertDist = sqrt((x1-Rx)^2+(y1-Ry)^2);

      if VertAngle > pi/2            %?????????????
         Quad2 = 1;
      elseif VertAngle < -pi/2
         Quad3 = 1;
      end
      if Quad2 && Quad3              %??????????
         break
      end
      if NumVert == 1  % First vertex- Set min and max
         MinAngle = VertAngle;
```

```
      MaxAngle = VertAngle;
      MinDist = VertDist;
      MaxDist = VertDist;

   else       % Find actual min and max values
      if VertAngle < MinAngle
         MinAngle = VertAngle;
      elseif VertAngle > MaxAngle
         MaxAngle = VertAngle;
      end
      if VertDist < MinDist
         MinDist = VertDist;
      elseif VertDist > MaxDist
         MaxDist = VertDist;
      end
   end
end

if Quad2 && Quad3    % If object is to the left of position, re-do calculations
   for NumVert = 1:VertPerObst(NumObst)
      x1 = OBST(2*NumObst-1, NumVert);
      y1 = OBST(2*NumObst, NumVert);

      VertAngle = atan2(y1-Ry,x1-Rx);
      if VertAngle < 0
         VertAngle = 2*pi + VertAngle;
      end

      VertDist = sqrt((x1-Rx)^2+(y1-Ry)^2);

      if NumVert == 1
         MinAngle = VertAngle;
         MaxAngle = VertAngle;
         MinDist = VertDist;
         MaxDist = VertDist;
      else
         if VertAngle < MinAngle
            MinAngle = VertAngle;
         elseif VertAngle > MaxAngle
            MaxAngle = VertAngle;
         end

         if VertDist < MinDist
            MinDist = VertDist;
         elseif VertDist > MaxDist
            MaxDist = VertDist;
         end
      end
   end

   if(Tgt_Angle < 0)
```

```
        Tgt_Angle = 2*pi + Tgt_Angle;
      end
    end

    % Now check to prove there is no Line of Sight

    % If past furthest vertex and within Object FOV
    if Tgt_Angle > MinAngle && Tgt_Angle < MaxAngle && Dist > MaxDist
      LOS = 0;
      if MinDist<CO_MD
        Closest_Obst_FOV=NumObst;
        CO_MD=MinDist;
      end
     % break
    % If within object's shadow
    elseif Tgt_Angle > MinAngle && Tgt_Angle < MaxAngle && Dist > MinDist && Dist < MaxDist
      % check for Intersections if within Object radii
      for NumVert = 1:VertPerObst(NumObst)
        x1 = OBST(2*NumObst-1, NumVert);
        x2 = OBST(2*NumObst-1, NumVert+1);
        y1 = OBST(2*NumObst, NumVert);
        y2 = OBST(2*NumObst, NumVert+1);
        if x2-x1==0
          x2=x2+.0001;
        end
        if y2-y1==0
          y2=y2+.0001;
        end
        VertSlope = (y2-y1)/(x2-x1);
        VertYIntercept = y1 - x1*VertSlope;

        if(VertSlope ~= Slope2Tgt)    % lines not parallel, therefore they intersect

          [xint,yint] = intersection(Slope2Tgt,TgtYIntercept,VertSlope,VertYIntercept);
          Dist2Intercept=sqrt((Rx-xint)^2+(Ry-yint)^2);
          if xint > min(x1,x2) && xint < max(x1,x2) && yint > min(y1,y2) && yint < max(y1,y2) &&
Dist > Dist2Intercept % intersection on obj border
            LOS = 0;
            if Dist2Intercept<CO_MD
              Closest_Obst_FOV=NumObst;
              CO_MD=Dist2Intercept;
            end
            break
          end
        end
      end
    end
end
```

Appendix L (Print_Environment.m)

```
global VertPerObst
global OBST
global Target
global Robot_Init

figure(1);
clf;
axis([0 15 0 15]);
hold on

for i = 1:NumObst
   xo = OBST(2*i-1,1:VertPerObst(i));          %Last space closes obstacle.
   yo = OBST(2*i,1:VertPerObst(i));
   fill(xo, yo, 'r');

end


[n,m]=size(Target);
for a=1:n
   plot(Target(a,1),Target(a,2),'g+')
end


[n,m]=size(Robot_Init);
Robot_Init=[Robot_Init zeros(n,1)];
for a=1:n
   plot(Robot_Init(a,1),Robot_Init(a,2),'b+')
end
```

Appendix M (Repel_From_Objectives.m)

```
function [ObjRepX ObjRepY]=Repel_From_Objectives(BotNumber)
% Determine Repulsive Potential Field From Other Robots

global NumTgts Robot Target RobotRad

SafeZone=.5;
ObjRepX=0;
ObjRepY=0;
for TgtNum=1:NumTgts
```

```
   Dist=sqrt((Target(TgtNum,1)-Robot(BotNumber,1))^2+(Target(TgtNum,2)-Robot(BotNumber,2))^2)-
2*RobotRad;
   Ang=atan2((Target(TgtNum,2)-Robot(BotNumber,2)),(Target(TgtNum,1)-Robot(BotNumber,1)));
   if Dist<SafeZone
      ObjRepX=ObjRepX-5*cos(Ang);
      ObjRepY=ObjRepY-5*sin(Ang);
   end
end
```

Appendix N (Repel_From_Obstacle.m)

```
function [ObstRepX ObstRepY]=Repel_From_Obstacle(BotNumber)
%Determines Repulsive Potential Field For Closest Obstacle

global OBST Robot VertPerObst RobotRad

SafeZone=.75;
ObstRepX=0;
ObstRepY=0;
[Dist2Obst ObstBearing Closest_Obst DistArray BearingArray]=Find_Closest_Obstacle(BotNumber);

for a=1:length(DistArray)

   if DistArray(a) < SafeZone
      ObstRepX = ObstRepX-cos(BearingArray(a))*(8/(DistArray(a)));
      ObstRepY = ObstRepY-sin(BearingArray(a))*(8/(DistArray(a)));
   end
end
```

Appendix O (Repel_From_Robots.m)

```
function [RobotRepX RobotRepY]=Repel_From_Robots(BotNumber)
% Determine Repulsive Potential Field From Other Robots

global NumBots Robot RobotRad

SafeZone=.50;
```

```
RobotRepX=0;
RobotRepY=0;
for BotNum=1:NumBots
   if BotNum~=BotNumber
      Dist=sqrt((Robot(BotNumber,1)-Robot(BotNum,1))^2+(Robot(BotNumber,2)-
Robot(BotNum,2))^2)-2*RobotRad;
      Ang=atan2((Robot(BotNum,2)-Robot(BotNumber,2)),(Robot(BotNum,1)-Robot(BotNumber,1)));
      if Dist<SafeZone
         RobotRepX=RobotRepX-5*cos(Ang);
         RobotRepY=RobotRepY-5*sin(Ang);
      end
   end
end
```

Appendix P (Task_Space_Controller.m)

```
function TaskVar = Task_Space_Controller()
%Determines all desired motion for secondary objectives.

global Robot Target NumTgts NumBots PosHistory

TaskVar=[];

for TgtNumber=1:1:NumTgts
   TaskVarR=[];
   for BotNumber=1:1:NumBots

      [ObstRepX ObstRepY]=Repel_From_Obstacle(BotNumber);     %Obstacle Repulsive Vector

      [RobotLOSX RobotLOSY]=Achieve_LOS(BotNumber,TgtNumber); %Move Around Obstacle
Vector

      [RobotRepX RobotRepY]=Repel_From_Robots(BotNumber);     %Robot Repulsive Vector

      [PastRepX PastRepY]=Avoid_Past(BotNumber);              %Prevents Local Minima To An Extent

      [ObjRepX ObjRepY]=Repel_From_Objectives(BotNumber);     %Prevents collision with Objective

      potX=ObstRepX+RobotLOSX+RobotRepX+PastRepX+ObjRepX;
      potY=ObstRepY+RobotLOSY+RobotRepY+PastRepY+ObjRepY;
      TaskVarR=[TaskVarR; potX; potY;0];
   end
   if TgtNumber==1
      TaskVar=TaskVarR;
   else
      TaskVar=TaskVar+TaskVarR;
   end
```

```
end
```

PHASE 2

Appendix a (Capability.m)

```
function C = Capability()     %Returns The Jacobian and Capability Values per Robot
global  M Opt_Pix dPix NumBots NumTgts Robot Target PixelsPerDM

   %This for statement cycles through the number of robots and calculates
   %the capability being delivered to each of the target locations.
Cb=[];
Ct=[];
LOS=1;

for TgtNumber=1:1:NumTgts
   for BotNumber=1:1:NumBots
      D(BotNumber)=sqrt((Robot(BotNumber,1)-Target(TgtNumber,1))^2+(Robot(BotNumber,2)-
Target(TgtNumber,2))^2)/PixelsPerDM;
      Tgt_Angle(BotNumber)=atan2(Robot(BotNumber,2)-Target(TgtNumber,2),Target(TgtNumber,1)-
Robot(BotNumber,1));
%
LOS=Line_Of_Sight(Robot(BotNumber,1),Robot(BotNumber,2),Target(TgtNumber,1),Target(TgtNumb
er,2));
      Diff_Angle=abs(Robot(BotNumber,8)-Tgt_Angle(BotNumber));
      while Diff_Angle>(2*pi)
         Diff_Angle=Diff_Angle-2*pi;
      end
      if Diff_Angle>pi
         Diff_Angle=2*pi-Diff_Angle;
      end
      if LOS==1 && Diff_Angle<(pi*.035/(8*Robot(BotNumber,5)))      %If the target is in the Field of
View
         if D<Robot(BotNumber,6)
```

```
            Cb(BotNumber)=sin((pi/2)*(D(BotNumber)/Robot(BotNumber,6)))^M*(10^(-
sin((Robot(BotNumber,8)-Tgt_Angle(BotNumber))/2)^4));
         else
            Cb(BotNumber)=sin((pi/2)*Robot(BotNumber,5)/(D(BotNumber)*dPix*Opt_Pix))*(10^(-
sin((Robot(BotNumber,8)-Tgt_Angle(BotNumber))/2)^4));
         end
      else
         Cb(BotNumber)=0;
      end
   end
   Ct=[Ct;Cb];
end
C=Ct;
```

Appendix b (Centroid.m)

```
function [Xc, Yc]=Centroids(Data)


Xc=0;
Yc=0;
for i = 1:320
   for j = 1:240
      if (Data(j,i) ==1)
         Xc = Xc + j;
         Yc = Yc + i;
      end
   end
end
Count=sum(sum(Data));
Xc=Xc/Count;
Yc=Yc/Count;
```

Appendix c (Controller.m)

```
%%%%%%%%%%%%% GLOBAL DECLARATIONS%%%%%%%%%%%%%%%%%%%%
clear;clc;
global dt M Opt_Pix dPix NumBots NumTgts Robot Target OBST VertPerObst
global RobotRad PosHistory DistArray s vid PixelsPerDM MOVIE

MOVIE=[];
figure(1);
```

```
clf;
axis([0 320 0 240]);
hold on
PixelsPerDM=24.5;
%%%%%%%%%%%%%% INITIALIZE ENVIRONMENT%%%%%%%%%%%%%%%%
vid=videoinput('winvideo');
preview(vid);
NumBots=input('How many robots are in the swarm?');

temp = 'Please Select Target Locations. (Double Click Final Target)';
disp(temp);
input('Press ENTER when ready');
[X,Y] = getpts;
Target=[X Y];
[n,m]=size(Target);
for a=1:n
   Target(a,2)=240-Target(a,2);
   plot(Target(a,1),Target(a,2),'g+')
end
[NumTgts b]=size(Target);          %Number of Targets in System
%s = Initialize_Robots(NumBots);
RunNumber=1;
Get_Environment(RunNumber);
for i = 1:NumBots
   Robot(i,5)=.035;
   Robot(i,7)=0;
   Robot(i,8)=Robot(i,3);
end
RunNumber=2;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%

%%%%%%%%%%%INITIALIZE ROBOT CHARACTERISTICS%%%%%%%%%%%

RobotRad=.4;
dt=.005;
M=5;
MIN_SPEED=.1;
MAX_SPEED=.5;
dPix=.0000856;       %Pixel width in m in the image frame.


Opt_Pix=400;         %Optimal Number of Pixels per Target Area
%[NumBots a]=size(Robot);           %Number of Robots in Swarm
%[NumTgts b]=size(Target);          %Number of Targets in System
for a=1:NumBots
   k=Robot(a,5)/(Opt_Pix*dPix); %Distance to Achieve Opt_Pix
   Robot(a,6)=k;
end
```

```
            %Robot Matrix Robot=[ X1 Y1 Theta1 DesiredHeading1 FocalLength1 k1 MaxSpeed1
CameraHeading1]
            %               X2 Y2 Theta2 DesiredHeading2 FocalLength2 k2 MaxSpeed2
CameraHeading2]
            %               ...]
            %Target Matrix Target=[X1 Y1
            %               X2 Y2
            %               ... ]
            %Obstacle Matrix OBST=[Ax1 Ax2 Ax3...
            %               Ay1 Ay2 Ay3...
            %               Bx1 Bx2 Bx3...
            %               By1 By2 By3...]
kp=1;
kt=1;        %Proportional Gain
Cd=[4.2];  %***TEMPORARY*** (just makes sure each bot achieves max cap. delivery)
NotDone=0;
CHist=[];
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% PosHistory=[];
Time=0;

while NotDone==0
%    PHT=[];
%    for B=1:1:NumBots
%        PHT=[PHT;Robot(B,1);Robot(B,2)];
%    end
%    PosHistory=[PosHistory PHT];
% %%%%%%%%%%%%%%%%%%%%%%%%%%% Calculate Jacobian and
Capability%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
   J=Jacobian();        %Returns a Jacobian Matrix for the system
   C=Capability();      %Returns the capability for each individual robot
   pJ=J'*((J*J')^(-1));         %Pseudo-Inverse for entire system
   CTotal=C(:,1);
   for i=2:1:NumBots
      CTotal=CTotal+C(:,i);                %Sums all of the individual capabilities
   end
   CHist=[CHist CTotal];

   for z=1:length(Cd)
      if CTotal(z)>=Cd(z)
         NotDone=NotDone+1;
      end
   end
   if NotDone==length(Cd)
      break;
   end
   NotDone=0;
```

```
%%%%%%%%%%%%%%%% SECONDARY CONTROLLER
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

   TaskVar=Task_Space_Controller();
   a=length(J);
   I=eye(a);


   qDot=pJ*(kp*(Cd-CTotal))
   x=(I-pJ*J)*TaskVar          %Desired Changes in X, Y AND Theta
   for a=1:NumBots
      Robot(a,10)=atan2(-qDot(3*a-1),qDot(3*a-2));
      Control_MAG=sqrt(qDot(3*a-1)^2+qDot(3*a-2)^2);
      Task_MAG=abs(x(3*a-1))+abs(x(3*a-2));
      if Task_MAG==0
         Task_MAG=1;
      end
      x(3*a-1)=Control_MAG*x(3*a-1)/Task_MAG;
      x(3*a-2)=Control_MAG*x(3*a-2)/Task_MAG;
   end
   x=kt*x
%    t1=atan2(-x(2),x(1))
   qDot=qDot+x;

   for a=1:NumBots
      Robot(a,4)=atan2(-qDot(3*a-1),qDot(3*a-2));
      t2=Robot(1,4);
      Robot(a,7)=sqrt(qDot(3*a-1)^2+qDot(3*a-2)^2);
      if Robot(a,7)>MAX_SPEED
         Robot(a,7)=MAX_SPEED;
      elseif Robot(a,7)<MIN_SPEED
         Robot(a,7)=MIN_SPEED;
      end
   end
   Move_Robot()

% %%%%%%%%%%%%%% MOVEMENT COMMANDS%%%%%%%%%%%%%%%%
   %Adjust Camera Heading Depending on qDot Value
   for BotNumber=1:1:NumBots
      Robot(BotNumber,8)=Robot(BotNumber,8)+100*qDot(3*BotNumber);
   end



%    CameraA=Robot(1,8)
   Get_Environment(RunNumber);
   CTotal;
   Robot(1,4);%
%
   Time=Time+dt;
```

```
%     input('Press Enter to Continue')

end



%%%%%%%%%FINAL VIEW%%%%%%%%%%%%%%%%%%%%%%%%%%%
I=getsnapshot(vid);
h = ones(5,5) / 50;
frame= imfilter(I,h);
yellowData = frame(:,:,1) >60 & frame(:,:,1)<130 & frame(:,:,2)>45 & frame(:,:,2) <115 & frame(:,:,3) >
20 & frame(:,:,3)<100;
%greenData = frame(:,:,1) <53 & frame(:,:,1) >35 & frame(:,:,2) >30 & frame(:,:,2) <85 &
frame(:,:,3)>15 & frame(:,:,3)<55;
redData = frame(:,:,1) >40 & frame(:,:,1) <80 & frame(:,:,2) >7 & frame(:,:,2) <45 & frame(:,:,3) > 7 &
frame(:,:,3) < 40;
yellowData=bwareaopen(yellowData,10);
%greenData=bwareaopen(greenData,75);
redData=bwareaopen(redData,30);
yellowData=imfill(yellowData,'holes');
%greenData=imfill(greenData,'holes');
redData=imfill(redData,'holes');
RED=bwlabel(redData);
YELLOW=bwlabel(yellowData);
%GREEN=bwlabel(greenData);
RED_Stats=regionprops(RED,'all');
YELLOW_Stats=regionprops(YELLOW,'all');
%GREEN_Stats=regionprops(GREEN,'all');
%%%%%%%%%ESTABLISH ROBOT X,Y LOCATIONS%%%%%%%%%
for a=1:NumBots
   TEMP1(a,1:2)=RED_Stats(a).Centroid;
   TEMP2(a,1:2)=YELLOW_Stats(a).Centroid;
end

for b=1:NumBots
   MaxDist=5;
   Done=0;
   while Done==0
      for e=1:NumBots
         DistTEMP=sqrt((Robot(b,1)-TEMP1(e,1))^2+(Robot(b,2)-TEMP1(e,2))^2);
         if DistTEMP<MaxDist
            Robot(b,1:2)=TEMP1(e,1:2);
            Rear(b,1:2)=TEMP2(e,1:2);
            Done=1;
            break;
         end
      end
      MaxDist=MaxDist+5;
   end
end
for c=1:NumBots
```

```
   for d=1:NumBots
      DistR2Y=sqrt((Robot(c,1)-Rear(d,1))^2+(Robot(c,2)-Rear(d,2))^2);
      if DistR2Y>14 && DistR2Y<18
         Robot(c,3)=atan2(Rear(d,2)-Robot(c,2),Robot(c,1)-Rear(d,1));
         break;
      end
   end
end

% Target(1,1:2)=GREEN_Stats(1).Centroid;

imshow(yellowData |  redData );
hold on;
for b=1:NumBots

plot(Robot(b,1),Robot(b,2),'r+',Rear(b,1),Rear(b,2),'y+',[Robot(b,1),Robot(b,1)+Robot(b,6)*PixelsPerDM
*cos(Robot(b,3))],[Robot(b,2),Robot(b,2)-Robot(b,6)*PixelsPerDM*sin(Robot(b,3))],'b',...
      [Robot(b,1),Robot(b,1)+Robot(b,6)*PixelsPerDM*cos(Robot(b,8))],[Robot(b,2),Robot(b,2)-
Robot(b,6)*PixelsPerDM*sin(Robot(b,8))],'y',Target(1,1),Target(1,2),'g+',...
       [Robot(b,1),Robot(b,1)+Robot(b,7)*10*PixelsPerDM*cos(Robot(b,10))],[Robot(b,2),Robot(b,2)-
Robot(b,7)*PixelsPerDM*10*sin(Robot(b,10))],'g',...
      [Robot(b,1),Robot(b,1)+Robot(b,7)*10*PixelsPerDM*cos(Robot(b,9))],[Robot(b,2),Robot(b,2)-
Robot(b,7)*PixelsPerDM*10*sin(Robot(b,9))],'r',...
     [Robot(b,1),Robot(b,1)+Robot(b,7)*10*PixelsPerDM*cos(Robot(b,4))],[Robot(b,2),Robot(b,2)-
Robot(b,7)*PixelsPerDM*10*sin(Robot(b,4))],'w');
end

%%%%%%%%%%STOP COMMAND%%%%%%%%%%%%%%
for j=1:5
   try
   fprintf(s(j),'D,0,0')
   catch
   end
end


%%%%%%DISPLAY CAPABILITY PLOT%%%%%%%%%%%%%%
T=0:dt:dt*(length(CHist)-1);
n=length(T);
CDisp=[];
for TgtNumber=1:1:NumTgts
   for a=1:1:n
      CDisp(a)=Cd(TgtNumber);
   end
      figure(TgtNumber+2)
      plot(T,CHist(TgtNumber,:),'r',T,CDisp,'b');
      title(['Capability at Target #', num2str(TgtNumber)]);
      hold on;
end
```

Appendix d (Get_Environment.m)

```
%%%%%%%%%%%%%%%%PseudoCode%%%%%%%%%%%%%%%%%%%%%
%   1.) Take Picture
%   2.) Locate Target Positions
%   3.) Locate Robot Positions and Orientation
%   4.) Return Robot Matrix and Target Matrix
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function Get_Environment(RunNumber)

global vid Robot Target NumBots NumTgts PixelsPerDM MOVIE

I=getsnapshot(vid);
h = ones(5,5) / 50;
frame=imfilter(I,h);
imshow(frame)
yellowData = frame(:,:,1) >60 & frame(:,:,1)<130 & frame(:,:,2)>45 & frame(:,:,2) <115 & frame(:,:,3) >
20 & frame(:,:,3)<100;
%greenData = frame(:,:,1) <53 & frame(:,:,1) >35 & frame(:,:,2) >30 & frame(:,:,2) <85 &
frame(:,:,3)>15 & frame(:,:,3)<55;
redData = frame(:,:,1) >40 & frame(:,:,1) <80 & frame(:,:,2) >7 & frame(:,:,2) <45 & frame(:,:,3) > 7 &
frame(:,:,3) < 40;
yellowData=bwareaopen(yellowData,15);
%greenData=bwareaopen(greenData,75);
redData=bwareaopen(redData,20);
yellowData=imfill(yellowData,'holes');
%greenData=imfill(greenData,'holes');
redData=imfill(redData,'holes');
RED=bwlabel(redData);
YELLOW=bwlabel(yellowData);
%GREEN=bwlabel(greenData);
RED_Stats=regionprops(RED,'all');
YELLOW_Stats=regionprops(YELLOW,'all');
%GREEN_Stats=regionprops(GREEN,'all');
%%%%%%%%ESTABLISH ROBOT X,Y LOCATIONS%%%%%%%%%
imshow(yellowData |  redData );

for a=1:NumBots
   TEMP1(a,1:2)=RED_Stats(a).Centroid;
   TEMP2(a,1:2)=YELLOW_Stats(a).Centroid;
end
if RunNumber==1
   Robot(:,1:2)=TEMP1(:,1:2);
   Rear(:,1:2)=TEMP2(:,1:2);
else
   for b=1:NumBots
```

```
      MaxDist=5;
      Done=0;
      while Done==0
         for e=1:NumBots
            DistTEMP=sqrt((Robot(b,1)-TEMP1(e,1))^2+(Robot(b,2)-TEMP1(e,2))^2);
            if DistTEMP<MaxDist
               Robot(b,1:2)=TEMP1(e,1:2);
               Rear(b,1:2)=TEMP2(e,1:2);
               Done=1;
               break;
            end
         end
         MaxDist=MaxDist+5;
      end
   end
end
for c=1:NumBots
   for d=1:NumBots
      DistR2Y=sqrt((Robot(c,1)-Rear(d,1))^2+(Robot(c,2)-Rear(d,2))^2);
      if DistR2Y>14 && DistR2Y<18
         Robot(c,3)=atan2(Rear(d,2)-Robot(c,2),Robot(c,1)-Rear(d,1));
         break;
      end
   end
end
if RunNumber==1
   Robot(:,8)=Robot(:,3);
   Robot(:,4)=Robot(:,3);
   Robot(:,9)=Robot(:,3);
   Robot(:,10)=Robot(:,3);
end

% Target(1,1:2)=GREEN_Stats(1).Centroid;

imshow(yellowData |  redData );
hold on;
for b=1:NumBots

plot(Robot(b,1),Robot(b,2),'r+',Rear(b,1),Rear(b,2),'y+',[Robot(b,1),Robot(b,1)+Robot(b,6)*PixelsPerDM
*cos(Robot(b,3))],[Robot(b,2),Robot(b,2)-Robot(b,6)*PixelsPerDM*sin(Robot(b,3))],'b',...
      [Robot(b,1),Robot(b,1)+Robot(b,6)*PixelsPerDM*cos(Robot(b,8))],[Robot(b,2),Robot(b,2)-
Robot(b,6)*PixelsPerDM*sin(Robot(b,8))],'y',Target(1,1),Target(1,2),'g+',...
      [Robot(b,1),Robot(b,1)+Robot(b,7)*10*PixelsPerDM*cos(Robot(b,10))],[Robot(b,2),Robot(b,2)-
Robot(b,7)*PixelsPerDM*10*sin(Robot(b,10))],'g',...
      [Robot(b,1),Robot(b,1)+Robot(b,7)*10*PixelsPerDM*cos(Robot(b,9))],[Robot(b,2),Robot(b,2)-
Robot(b,7)*PixelsPerDM*10*sin(Robot(b,9))],'r',...
      [Robot(b,1),Robot(b,1)+Robot(b,7)*10*PixelsPerDM*cos(Robot(b,4))],[Robot(b,2),Robot(b,2)-
Robot(b,7)*PixelsPerDM*10*sin(Robot(b,4))],'w');
end
F=getframe;
MOVIE=[MOVIE F];
```

Appendix e (Initialize_Robots.m)

```
function s=Initialize_Robots(NumBots)


for i = 1:NumBots
   if i==5
      disp('Capturing serial port 10 ...');
      s(i) = serial('COM10');
      set(s(i), 'BaudRate', 115200);
      disp('Opening serial port 10 ...');
      fopen(s(i));
      while(get(s(i), 'BytesAvailable')) >0
         idn = fread(s(i), 1)
      end
   else
      disp(['Capturing serial port ', num2str(3+i), ' ...']);
      s(i) = serial(['COM', num2str(3+i)]);
      set(s(i), 'BaudRate', 115200);
      disp(['Opening serial port ', num2str(3+i), ' ...']);
      fopen(s(i));
      while(get(s(i), 'BytesAvailable')) >0
         idn = fread(s(i), 1)
      end
   end
end

disp('Ready');
```

Appendix f (Jacobian.m)

```
function J=Jacobian()     %Returns The Jacobian and Capability Values per Robot
global  M f Opt_Pix dPix NumBots NumTgts Robot Target PixelsPerDM

   %This for statement cycles through the number of robots and calculates
   %the partial derivative for movement in the X and Y direction as well
   %as a change in Theta.  These are then compiled into 1 Jacobian for
   %the entire system.
Jb=[];
```

```
Jt=[];
for TgtNumber=1:1:NumTgts
   for BotNumber=1:1:NumBots
      D(BotNumber)=sqrt((Robot(BotNumber,1)-Target(TgtNumber,1))^2+(Robot(BotNumber,2)-
Target(TgtNumber,2))^2)/PixelsPerDM;
      Tgt_Angle(BotNumber)=atan2(Robot(BotNumber,2)-Target(TgtNumber,2),Target(TgtNumber,1)-
Robot(BotNumber,1));
      if D<Robot(BotNumber,6)
         dCdX=M/4*sin(1/2*pi*D(BotNumber)/Robot(BotNumber,6))^(M-1)*(10^(-
sin((Robot(BotNumber,8)-Tgt_Angle(BotNumber))/2)^4))*...

cos(1/2*pi*D(BotNumber)/Robot(BotNumber,6))*pi/D(BotNumber)/Robot(BotNumber,6)*2*(Robot(Bo
tNumber,1)-Target(TgtNumber,1));
         dCdY=M/4*sin(1/2*pi*D(BotNumber)/Robot(BotNumber,6))^(M-1)*(10^(-
sin((Robot(BotNumber,8)-Tgt_Angle(BotNumber))/2)^4))*...

cos(1/2*pi*D(BotNumber)/Robot(BotNumber,6))*pi/D(BotNumber)/Robot(BotNumber,6)*2*(Robot(Bo
tNumber,2)-Target(TgtNumber,2));
         dCdTheta=-2*(sin(pi*D(BotNumber)/(2*Robot(BotNumber,6)))^M)*cos((Robot(BotNumber,8)-
Tgt_Angle(BotNumber))/2)*...
            (sin((Robot(BotNumber,8)-Tgt_Angle(BotNumber))/2)^3)*log(10)*10^(-
sin((Robot(BotNumber,8)-Tgt_Angle(BotNumber))/2)^4);
      else
         dCdX=2*(Robot(BotNumber,1)-Target(TgtNumber,1))*(-
pi/4*Robot(BotNumber,5)/(dPix*Opt_Pix)*D(BotNumber)^(-
3))*cos(pi/2*Robot(BotNumber,5)/(D(BotNumber)*...
            dPix*Opt_Pix))*10^(-(sin((Robot(BotNumber,8)-Tgt_Angle(BotNumber))/2)^4));
         dCdY=2*(Robot(BotNumber,2)-Target(TgtNumber,2))*(-
pi/4*Robot(BotNumber,5)/(dPix*Opt_Pix)*D(BotNumber)^(-
3))*cos(pi/2*Robot(BotNumber,5)/(D(BotNumber)*...
            dPix*Opt_Pix))*10^(-(sin((Robot(BotNumber,8)-Tgt_Angle(BotNumber))/2)^4));
         dCdTheta=-
2*sin((pi/2)*Robot(BotNumber,5)/(D(BotNumber)*dPix*Opt_Pix))*cos((Robot(BotNumber,8)-
Tgt_Angle(BotNumber))/2)*...
            (sin((Robot(BotNumber,8)-Tgt_Angle(BotNumber))/2)^3)*log(10)*10^(-
sin((Robot(BotNumber,8)-Tgt_Angle(BotNumber))/2)^4);
      end
      Jb=[Jb dCdX dCdY dCdTheta];
   end
   Jt=[Jt;Jb];
   Jb=[];
end
J = Jt;
```

Appendix g (Move_Robot.m)

```
function  Move_Robot()

global Robot Target s PixelsPerDM NumBots

%%%%%%%%%%%%%%%%%Robot Specifications%%%%%%%%%%%%%
d=.053;
L=.03;
r=.008;
k=.1;
Ktheta=100;
Wheel_Vel=[];
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%GET WHEEL VELOCITIES%%%%%%%%%%%%%%
for i=1:NumBots

   Current_Heading=Robot(i,3);
   Desired_Heading=Robot(i,4);
   Desired_Velocity=Robot(i,7);
   if Current_Heading<0
      Current_Heading=Current_Heading+2*pi;
   end
   if Desired_Heading<0
      Desired_Heading=Desired_Heading+2*pi;
   end




   Diff_Angle=Desired_Heading-Current_Heading;
   if Diff_Angle>pi
      Diff_Angle=Diff_Angle-2*pi;
   elseif Diff_Angle<-180
      Diff_Angle=Diff_Angle+2*pi;
   end



   %%%%DETERMINE DIRECTION TO TURN%%%%
%    if Diff_Angle<0
%       'Turning Right...'
%    else
%       'Turning Left...'
%    end

   Wheel_Vel_Temp=[2*k/r -1*Ktheta*L*d/r ; 2*k/r
Ktheta*L*d/r]*[Desired_Velocity*cos(Diff_Angle);Desired_Velocity*sin(Diff_Angle)];

   %%%%%%%%Thresholds Maximum Wheel Velocity%%%%%%%%%%
   if Wheel_Vel_Temp(1) >1 || Wheel_Vel_Temp(1) <-1 || Wheel_Vel_Temp(2)>1 ||
Wheel_Vel_Temp(2)<-1
```

```
        TotWV=abs(Wheel_Vel_Temp(1))+abs(Wheel_Vel_Temp(2));
        Wheel_Vel_Temp(1)=Wheel_Vel_Temp(1)/TotWV;
        Wheel_Vel_Temp(2)=Wheel_Vel_Temp(2)/TotWV;
    end
    Wheel_Vel=[Wheel_Vel; Wheel_Vel_Temp];
while(get(s(i), 'BytesAvailable')) >0
    idn = fread(s, 1);
end
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%$%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%

Wheel_Vel;



%%%%%%%%%%%%SEND WHEEL COMMANDS%%%%%%%%%%%%%%%
for j=1:NumBots
    try
    fprintf(s(j),'D,%d,%d\n',round([Wheel_Vel(2*j-1),Wheel_Vel(2*j)]))
    catch
    end
%     pause(1)
%     try
%     fprintf(s(j),'D,0,0')
%     catch
%     end
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Appendix h (Repel_From_Objectives.m)

```
function [ObjRepX ObjRepY]=Repel_From_Objectives(BotNumber)
% Determine Repulsive Potential Field From Other Robots

global NumTgts Robot Target RobotRad

SafeZone=.5;
ObjRepX=0;
ObjRepY=0;
for TgtNum=1:NumTgts
    Dist=sqrt((Target(TgtNum,1)-Robot(BotNumber,1))^2+(Target(TgtNum,2)-Robot(BotNumber,2))^2)-
2*RobotRad;
    Ang=atan2((Target(TgtNum,2)-Robot(BotNumber,2)),(Target(TgtNum,1)-Robot(BotNumber,1)));
    if Dist<SafeZone
        ObjRepX=ObjRepX-5*cos(Ang);
```

```
      ObjRepY=ObjRepY-5*sin(Ang);
   end
end
```

Appendix i (Repel_From_Obstalce.m)

```
function [ObstRepX ObstRepY]=Repel_From_Obstacle(BotNumber)
%Determines Repulsive Potential Field For Closest Obstacle

global Robot s
ObstRepX=0;
ObstRepY=0;
while(get(s(BotNumber), 'BytesAvailable')) >0
   idn = fread(s(BotNumber), 1);
end
try
   fprintf(s(BotNumber), 'N\n');
catch
end
while(get(s(BotNumber), 'BytesAvailable'))<20
end
done = 0;
I = [];
K = [];
while(~done)
   idn = fread(s(BotNumber), 1);
   if (idn == 13)
      idn = fread(s(BotNumber), 1);
      if (idn == 10)
         done = 1;
      end
   else
      if (idn ~= 44)
         I = [I, idn];
      else
         val = str2num(char(I));
         I = [];
         K = [K, val];
      end
   end
end
val = str2num(char(I));
I = [];
K = [K, val]
while(get(s(BotNumber), 'BytesAvailable')) >0
   idn = fread(s(BotNumber), 1);
```

```
end

for i=1:length(K)
   if K(i) > 170
      if i==1
         ObstRepX=cos(-pi/2);
         ObstRepY=sin(-pi/2);
      elseif i==2
         ObstRepX=ObstRepX+cos(-135*pi/180);
         ObstRepY=ObstRepY+sin(-135*pi/180);
      elseif i==3
         ObstRepX=ObstRepX+cos(-170*pi/180);
         ObstRepY=ObstRepY+sin(-170*pi/180);
      elseif i==4
         ObstRepX=ObstRepX+cos(170*pi/180);
         ObstRepY=ObstRepY+sin(170*pi/180);
      elseif i==5
         ObstRepX=ObstRepX+cos(135*pi/180);
         ObstRepY=ObstRepY+sin(135*pi/180);
      elseif i==6
         ObstRepX=ObstRepX+cos(90*pi/180);
         ObstRepY=ObstRepY+sin(90*pi/180);
%       elseif i==7
%          ObstRepX=ObstRepX+cos(10*pi/180);
%          ObstRepY=ObstRepY-sin(10*pi/180);
%       elseif i==8
%          ObstRepX=ObstRepX+cos(-10*pi/180);
%          ObstRepY=ObstRepY-sin(-10*pi/180);
      end
   end
end
```

Appendix j (Repel_From_Robots.m)

```
function [RobotRepX RobotRepY]=Repel_From_Robots(BotNumber)
% Determine Repulsive Potential Field From Other Robots

global NumBots Robot PixelsPerDM

SafeZone=1.5;
RobotRepX=0;
RobotRepY=0;
for BotNum=1:NumBots
   if BotNum~=BotNumber
      Dist=sqrt((Robot(BotNumber,1)-Robot(BotNum,1))^2+(Robot(BotNumber,2)-
Robot(BotNum,2))^2)/PixelsPerDM;
      Ang=atan2((Robot(BotNum,2)-Robot(BotNumber,2)),(Robot(BotNum,1)-Robot(BotNumber,1)));
```

```
      if Dist<SafeZone
         RobotRepX=RobotRepX-4*cos(Ang);
         RobotRepY=RobotRepY-4*sin(Ang);
      end
   end
end
```

Appendix k (RobotCentroids.m)

```
function [orientation, X_Center, Y_Center]=Centroids(RedData,YellowData)

Xc_Red=0;
Yc_Red=0;
Xc_Yellow=0;
Yc_Yellow=0;
for i = 1:320
   for j = 1:240
      if (RedData(j,i) ==1)
         Xc_Red = Xc_Red + j;
         Yc_Red = Yc_Red + i;
      end
      if (YellowData(j,i) ==1)
         Xc_Yellow = Xc_Yellow + j;
         Yc_Yellow = Yc_Yellow + i;
      end
   end
end
RedCount=sum(sum(RedData));
Xc_Red=Xc_Red/RedCount;
Yc_Red=Yc_Red/RedCount;
YellowCount=sum(sum(YellowData));
Xc_Yellow=Xc_Yellow/YellowCount;
Yc_Yellow=Yc_Yellow/YellowCount;
orientation=atan2(Xc_Yellow-Xc_Red,Yc_Red-Yc_Yellow);
X_Center=Xc_Red;
Y_Center=Yc_Red;
```

Appendix l (Swarm_Controller.m)

```
global dt M Opt_Pix dPix NumBots NumTgts Robot Target OBST VertPerObst
global RobotRad PosHistory DistArray Robot_Init
```

```
%%%%%ESTABLISH ENVIRONMENT AND INITIALIZATIONS%%%%%%%
Create_Environment();

Robot=Robot_Init;

Robot(1,5)=.035;
Robot(2,5)=.035;
Robot(3,5)=.035;
Robot(4,5)=.035;

Robot(1,7)=30;
Robot(2,7)=30;
Robot(3,7)=30;
Robot(4,7)=30;


dt=.005;
M=5;
MIN_SPEED=8;
dPix=.0000856;          %Pixel width in m in the image frame.

Opt_Pix=400;            %Optimal Number of Pixels per Target Area
[NumBots a]=size(Robot);              %Number of Robots in Swarm
[NumTgts b]=size(Target);             %Number of Targets in System
for a=1:NumBots
   k=Robot(a,5)/(Opt_Pix*dPix); %Distance to Achieve Opt_Pix
   Robot(a,6)=k;
end

                %Robot Matrix Robot=[ X1 Y1 Theta1 DesiredHeading1 FocalLength1 k1 MaxSpeed1]
                %                    X2 Y2 Theta2 DesiredHeading2 FocalLength2 k2 MaxSpeed2]
                %                    ...]
                %Target Matrix Target=[X1 Y1
                %                      X2 Y2
                %                      ... ]
                %Obstacle Matrix OBST=[Ax1 Ax2 Ax3...
                %                      Ay1 Ay2 Ay3...
                %                      Bx1 Bx2 Bx3...
                %                      By1 By2 By3...]
kp=1;
kt=5;           %Proportional Gain
Cd=[3.95];   %***TEMPORARY*** (just makes sure each bot achieves max cap. delivery)
NotDone=0;
CHist=[];
Alert='MAX REACHED';
Alert2='MIN REACHED';

PosHistory=[];
Time=0;
while NotDone==0
   PHT=[];
```

```
for B=1:1:NumBots
   PHT=[PHT;Robot(B,1);Robot(B,2)];
end
PosHistory=[PosHistory PHT];

J=Jacobian();        %Returns a Jacobian Matrix for the system
C=Capability();      %Returns the capability for each individual robot
pJ=J'*((J*J')^(-1));          %Pseudo-Inverse for entire system
CTotal=C(:,1);
for i=2:1:NumBots
   CTotal=CTotal+C(:,i);                  %Sums all of the individual capabilities
end
CHist=[CHist CTotal];

for z=1:length(Cd)
   if CTotal(z)>=Cd(z)
      NotDone=NotDone+1;
   end
end
if NotDone==length(Cd)
   break;
end
NotDone=0;
TaskVar=Task_Space_Controller();
a=length(J);
I=eye(a);



qDot=pJ*(kp*(Cd-CTotal))+kt*(I-pJ*J)*TaskVar;          %Desired Changes in X, Y AND Theta


for a=1:NumBots
   DesiredHeading=atan2(qDot(3*a-1),qDot(3*a-2));
   Speed=sqrt(qDot(3*a-1)^2+qDot(3*a-2)^2);
   Robot(a,4)=DesiredHeading;
   if Speed>Robot(a,7)
      qDot(3*a-2)=Robot(a,7)*cos(DesiredHeading);
      qDot(3*a-1)=Robot(a,7)*sin(DesiredHeading);
   elseif Speed<MIN_SPEED
      qDot(3*a-2)=MIN_SPEED*cos(DesiredHeading);
      qDot(3*a-1)=MIN_SPEED*sin(DesiredHeading);
   end
end
%*****HARD CODE MUST BE CHANGED DEPENDING ON NUMBER OF ROBOTS******
qDot2=[qDot(1:3)'; qDot(4:6)'; qDot(7:9)'; qDot(10:12)'];   %Changes qDot from a 12x1 Matrix to a
4x3 matrix corresponding to Robot



%Adjust Robot X,Y and Theta Values Depending on qDot Value
```

```
   for BotNumber=1:1:NumBots
      Robot(BotNumber,1)=Robot(BotNumber,1)+qDot2(BotNumber,1)*dt;
      Robot(BotNumber,2)=Robot(BotNumber,2)+qDot2(BotNumber,2)*dt;
      Robot(BotNumber,3)=Robot(BotNumber,3)+3*qDot2(BotNumber,3)*dt;
   end

   %Plot Robot Locations with Orientation
   figure(1);
   for TgtNumber=1:1:NumTgts
      for BotNumber=1:1:NumBots
         axis([0 15 0 15]);
         hold on;

plot(Robot(BotNumber,1),Robot(BotNumber,2),'bo',Target(TgtNumber,1),Target(TgtNumber,2),'g+',[Ro
bot(BotNumber,1),...

Robot(BotNumber,1)+Robot(BotNumber,6)*cos(Robot(BotNumber,3))],[Robot(BotNumber,2),Robot(Bo
tNumber,2)+Robot(BotNumber,6)*sin(Robot(BotNumber,3))],'y');%,...
%
[Robot(BotNumber,1),Robot(BotNumber,1)+Robot(BotNumber,6)*cos(Robot(BotNumber,4))],[Robot(B
otNumber,2),Robot(BotNumber,2)+Robot(BotNumber,6)*sin(Robot(BotNumber,4))],'r');

      end
   end
   figure(2);
   hold off;
   axis([0 3 0 4])
   bar([CTotal Cd],'group')
   hold on;

   Time=Time+dt;

end

%Plot Robot Locations with Orientation
   figure(1);
   for TgtNumber=1:1:NumTgts
      for BotNumber=1:1:NumBots
         axis([0 15 0 15]);
         hold on;

plot(Robot(BotNumber,1),Robot(BotNumber,2),'bo',Target(TgtNumber,1),Target(TgtNumber,2),'g+',[Ro
bot(BotNumber,1),...

Robot(BotNumber,1)+Robot(BotNumber,6)*cos(Robot(BotNumber,3))],[Robot(BotNumber,2),Robot(Bo
tNumber,2)+Robot(BotNumber,6)*sin(Robot(BotNumber,3))],'y');%,...
%
[Robot(BotNumber,1),Robot(BotNumber,1)+Robot(BotNumber,6)*cos(Robot(BotNumber,4))],[Robot(B
otNumber,2),Robot(BotNumber,2)+Robot(BotNumber,6)*sin(Robot(BotNumber,4))],'r');

      end
```

```
   end
   figure(2);
   hold off;
   axis([0 3 0 4])
   bar([CTotal Cd],'group')
   hold on;

disp('DONE')

T=0:dt:dt*(length(CHist)-1);
n=length(T);
CDisp=[];
for TgtNumber=1:1:NumTgts
   for a=1:1:n
      CDisp(a)=Cd(TgtNumber);
   end
      figure(TgtNumber+2)
      plot(T,CHist(TgtNumber,:),'r',T,CDisp,'b');
      title(['Capability at Target #', num2str(TgtNumber)]);
      hold on;
end

figure(NumTgts+3);
axis([0 15 0 15]);
hold on
for i = 1:NumObst
   xo = OBST(2*i-1,1:VertPerObst(i));          %Last space closes obstacle.
   yo = OBST(2*i,1:VertPerObst(i));
   fill(xo, yo, 'r');

end
[n,m]=size(Target);
for a=1:n
   plot(Target(a,1),Target(a,2),'g+')
end
for BotNumber=1:1:NumBots

plot(Robot(BotNumber,1),Robot(BotNumber,2),'bo',Target(TgtNumber,1),Target(TgtNumber,2),'g+',[Ro
bot(BotNumber,1),...

Robot(BotNumber,1)+Robot(BotNumber,6)*cos(Robot(BotNumber,3))],[Robot(BotNumber,2),Robot(Bo
tNumber,2)+Robot(BotNumber,6)*sin(Robot(BotNumber,3))],'y');%,...
%
[Robot(BotNumber,1),Robot(BotNumber,1)+Robot(BotNumber,6)*cos(Robot(BotNumber,4))],[Robot(B
otNumber,2),Robot(BotNumber,2)+Robot(BotNumber,6)*sin(Robot(BotNumber,4))],'r');
   hold on;
end
```

Appendix m (Task_Space_Controller.m)

```
function TaskVar = Task_Space_Controller()
%Determines all desired motion for secondary objectives.

global Robot Target NumTgts NumBots PosHistory PixelsPerDM

TaskVar=[];

for TgtNumber=1:1:NumTgts
   TaskVarR=[];
   for BotNumber=1:NumBots

      [ObstRepX ObstRepY]=Repel_From_Obstacle(BotNumber);     %Obstacle Repulsive Vector

%        [RobotLOSX RobotLOSY]=Achieve_LOS(BotNumber,TgtNumber); %Move Around Obstacle
Vector
%
      [RobotRepX RobotRepY]=Repel_From_Robots(BotNumber);     %Robot Repulsive Vector
%
%        [PastRepX PastRepY]=Avoid_Past(BotNumber);             %Prevents Local Minima To An Extent
%
%        [ObjRepX ObjRepY]=Repel_From_Objectives(BotNumber);     %Prevents collision with
Objective

      potX=ObstRepX+RobotRepX;%+ObjRepX;
      potY=ObstRepY+RobotRepY;%+ObjRepY;
      PotMag=sqrt(potX^2+potY^2);
      RepBearing=atan2(ObstRepY,ObstRepX);
      TSBearing=Robot(BotNumber,3)+RepBearing;
      Robot(BotNumber,9)=TSBearing;
      potX2=PotMag*cos(TSBearing);
      potY2=-PotMag*sin(TSBearing);
      TaskVarR=[TaskVarR; potX2; potY2;0];
   end
   if TgtNumber==1
      TaskVar=TaskVarR;
   else
      TaskVar=TaskVar+TaskVarR;
   end
end
```

Appendix n (VisionTest.m)

```
global R1_Orient DirToTarget DistToTarget NumBots
vid=videoinput('winvideo');
```

```
preview(vid);
%DISTANT UNITS IS Centimeters!!%%%%

PixelsPerDM=25;
AngPerCount=.175;
NumBots=5;
NOT_DONE=true;
I=getsnapshot(vid);
h = ones(5,5) / 50;
frame=imfilter(I,h);
imshow(frame)
yellowData = frame(:,:,1) >60 & frame(:,:,1)<130 & frame(:,:,2)>45 & frame(:,:,2) <115 & frame(:,:,3) >
20 & frame(:,:,3)<100;
%greenData = frame(:,:,1) <53 & frame(:,:,1) >35 & frame(:,:,2) >30 & frame(:,:,2) <85 &
frame(:,:,3)>15 & frame(:,:,3)<55;
redData = frame(:,:,1) >40 & frame(:,:,1) <80 & frame(:,:,2) >7 & frame(:,:,2) <45 & frame(:,:,3) > 7 &
frame(:,:,3) < 40;
yellowData=bwareaopen(yellowData,15);
%greenData=bwareaopen(greenData,75);
redData=bwareaopen(redData,20);
yellowData=imfill(yellowData,'holes');
%greenData=imfill(greenData,'holes');
redData=imfill(redData,'holes');
RED=bwlabel(redData);
YELLOW=bwlabel(yellowData);
%GREEN=bwlabel(greenData);
RED_Stats=regionprops(RED,'all');
YELLOW_Stats=regionprops(YELLOW,'all');
%GREEN_Stats=regionprops(GREEN,'all');
%%%%%%%%%ESTABLISH ROBOT X,Y LOCATIONS%%%%%%%%%
imtool(frame)
for a=1:NumBots
   Robot(a,1:2)=RED_Stats(a).Centroid;
   temp(a,1:2)=YELLOW_Stats(a).Centroid;
end
for c=1:NumBots
   for d=1:NumBots
      DistR2Y=sqrt((Robot(c,1)-temp(d,1))^2+(Robot(c,2)-temp(d,2))^2);
      if DistR2Y>14 && DistR2Y<19
         Robot(c,3)=atan2(temp(d,2)-Robot(c,2),Robot(c,1)-temp(d,1));
         'Match'
         c
         d
         break;
      end
   end
end


%Target(1,1:2)=GREEN_Stats(1).Centroid;
% DirToTarget=atan2(R1Xc-Tx,Ty-R1Yc)*180/pi;
```

```
DistToTarget=sqrt((Robot(1,1)-temp(1,1))^2+(Robot(1,2)-temp(1,2))^2)
PixelsPerDM=DistToTarget/.65
DistToTarget=DistToTarget/PixelsPerDM;
imshow(yellowData | redData );
hold on;
for b=1:NumBots

plot(Robot(b,1),Robot(b,2),'r+',temp(b,1),temp(b,2),'y+',[Robot(b,1),Robot(b,1)+100*cos(Robot(b,3))],[R
obot(b,2),Robot(b,2)-100*sin(Robot(b,3))],'b');%,Target(1,1),Target(1,2),'g+');
end
```

Endnotes

[1] Dunbar, T., Esposito, J., "**Artificial Potential Field Controllers for Robust Communications in a Network of Swarm Robots"** *Proceedings of the Thirty-Seventh Southeastern Symposium on System Theory*, 2005.
[2] Bishop, B.E., "**On the use of redundant manipulator techniques for control of platoons of cooperating robotic vehicles"** *IEEE Transactions on Systems, Man, and Cybernetics, Part A,* Sept. 2003.
[3] Goldbeck, J., "**Evolving optimal parameters for swarm control"** *NASA/DoD Conference on Evolvable Hardware,* 2002.
[4] Goldbeck, J., "**Evolving optimal parameters for swarm control"** *NASA/DoD Conference on Evolvable Hardware,* 2002
[5] Goldbeck, J., "**Evolving optimal parameters for swarm control"** *NASA/DoD Conference on Evolvable Hardware,* 2002
[6] Yong Chye Tan; Bishop, B.E., "**Combining Classical and Behavior-Based Control for Swarms of Cooperating Vehicles"** *Proceedings of the 2005 IEEE International Conference on Robotics and Autonomation*, 2005
[7] Bishop, B.E., "**On the use of redundant manipulator techniques for control of platoons of cooperating robotic vehicles"** *IEEE Transactions on Systems, Man, and Cybernetics, Part A,* Sept. 2003.
[8] Bishop, B.E., "**Control of Platoons of Nonholonomic Vehicles Using Redundant Manipulator Analogs"** *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*, 2005
[9] B. E. Bishop et.al., **ES452 Course Notes**, United States Naval Academy, 2006.
[10] Barnes, L.; Alvis, W.; Fields, M.; Valavanis, K.; Moreno, W., "**Heterogeneous Swarm Formation Control Using Bivariate Normal Functions to Generate Potential Fields"** *IEEE Workshop on Distributed Intelligent Systems: Collective Intelligence and Its Applications*, 2006
[11] Yong Chye Tan; Bishop, B.E., "**Combining Classical and Behavior-Based Control for Swarms of Cooperating Vehicles"** *Proceedings of the 2005 IEEE International Conference on Robotics and Autonomation*, 2005
[12] Khepera II Mobile Robot User Manual
[13] Star Key 2.0 USB Long Range Bluetooth Dongle Manual
[14] Khepera II Mobile Robot User Manual
[15] Khepera II Mobile Robot User Manual
[16] Khepera II Mobile Robot User Manual
[17] Khepera II Mobile Robot User Manual
[18] Khepera II Mobile Robot User Manual